

## مقدمه

سیستم های کامپیوترا در حال تحول هستند. از سال ۱۹۴۵، که آغاز دوران کامپیوتراهای امروزی محسوب می شود، تا حدود سال های ۱۹۸۵ کامپیوتراها هم بزرگ بودند و هم گران. بطوریکه یک مینی کامپیوترا حداقل هزاران دلار قیمت داشت. در نتیجه، اغلب سازمانها فقط تعداد انگشت شماری کامپیوترا در اختیار داشتند و به دلیل عدم امکان اتصال، همین کامپیوتراها هم مستقل از یکدیگر کار می کردند. اما تقریباً از اواسط دهه هشتاد میلادی، دو پیشرفت در فن آوری باعث تغییر شرایط شد. اولین تغییر مربوط به توسعه ریزپردازنده های<sup>۱</sup> قدرتمند بود. در ابتدا، این ریزپردازنده ها ماشین های ۸ بیتی بودند، اما کمی بعدتر پردازنده های ۳۲ و ۶۴ بیتی هم متداول شدند. قدرت محاسبه بسیاری از این ریزپردازنده ها به اندازه کامپیوتراهای بزرگ<sup>۲</sup> اما با کسری از قیمت آنها بود.

میزان رشدی که طی نیم قرن گذشته در عرصه فن آوری کامپیوتراها رخ داده، واقعاً مبهوت کننده و در مقایسه با صنایع دیگر، کاملاً بی سابقه است. بطوریکه از ماشین هایی با قیمت ۱۰ میلیون دلار که قادر به اجرای فقط یک دستورالعمل در عرض یک ثانیه بودند، امروزه به ماشین هایی رسیده ایم که ۱۰۰۰ دلار قیمت داشته و در عوض قادر به اجرای یک میلیارد دستورالعمل در واحد ثانیه بوده و نسبت کارآیی به قیمت آنها<sup>۳</sup> ۱۰ برابر شده است. اگر ماشین ها هم در این فاصله زمانی با همین سرعت رشد می کردند، امروز رولزرویس فقط یک دلار قیمت داشت و در یک میلیارد مایل فقط حدود چهار لیتر بنزین مصرف می کرد.

پیشرفت دوم مربوط به پیدایش شبکه های کامپیوترا پرسرعت بود. شبکه های محلی<sup>۴</sup> یا LAN ها باعث می شوند تا صدها ماشین در یک ساختمان طوری به هم متصل شوند که بتوان مقادیر کمی از اطلاعات را در عرض تقریباً چند میلی ثانیه بین آنها انتقال داد. انتقال مقادیر بزرگ تر داده ها در بین ماشین هایی با سرعت ۱۰۰ میلیون تا ۱۰ میلیارد بیت بر ثانیه امکان پذیر است. شبکه های گسترده<sup>۵</sup> یا WAN ها امکان اتصال میلیونها ماشین در سرتاسر جهان با سرعت متغیر از ۶۴ کیلوبیت در ثانیه (Kbps) تا گیگابیت در ثانیه را فراهم کرده است.

در اثر تمامی این فن آوری ها، امروزه در کنار هم قرار دادن سیستم های محاسباتی متشكل از تعداد زیادی کامپیوترا که بوسیله یک شبکه کامپیوترا پرسرعت بهم متصل شده باشند، امروزه نه تنها امکان پذیر، بلکه آسان هم شده است. این شبکه های کامپیوترا یا به اصطلاح سیستم های توزیعی<sup>۶</sup> با

<sup>1</sup> microprocessor

<sup>2</sup> mainframe

<sup>3</sup> local-area networks

<sup>4</sup> wide-area networks

<sup>5</sup> distributed systems

سیستم های متتمرکز<sup>۶</sup> ( یا سیستم های تک پردازشگری<sup>۷</sup> ) قدیمی که فقط از یک کامپیوتر، دستگاه های جانبی و احتمالاً پایانه های دور<sup>۸</sup> تشکیل می شدند، متفاوت است.

### ۱-۱-تعریف سیستم های توزیعی

در منابع مختلف تعاریف مختلفی هم برای سیستم های توزیعی ارائه شده است. اما هیچیک نه کامل است و نه با دیگری همخوانی دارد. برای اهداف ما در این کتاب ، ارائه یک تعریف تقریبی هم کفايت می کند :

سیستم توزیعی در واقع مجموعه ای از کامپیوترهای مستقل<sup>۹</sup> است که برای کاربر خود ، مانند یک سیستم منسجم<sup>۱۰</sup> و منفرد<sup>۱۱</sup> به نظر می رسد.

این تعریف نکته های مختلف و مهمی در خود دارد. اول آنکه، یک سیستم توزیعی از مؤلفه های (یعنی، کامپیوترهای) خود مختار تشکیل می شود . نکته دوم اینکه، کاربران (چه افراد و چه برنامه ها) تصور می کنند که با یک سیستم منفرد کار می کنند. به این معنا که مؤلفه های خود مختار<sup>۱۲</sup> بایستی به روشی با هم همکاری<sup>۱۳</sup> کنند. نحوه ایجاد این تشریک مساعی هسته سیستم های توزیعی را تشکیل می دهد. توجه داشته باشید که هیچ فرضی راجع به نوع کامپیوترها صورت نمی گیرد. اما در مجموع، حتی در داخل یک سیستم منفرد هم، انواع کامپیوترها، از کامپیوترهای بزرگ کارآ گرفته تا گره های کوچک در شبکه های حسگر، به چشم می خورد. به همین ترتیب، هیچ فرضی راجع به نحوه اتصال کامپیوترها صورت نمی گیرد. در ادامه فصل راجع به این جنبه های مختلف بحث خواهیم کرد.

شاید مفیدتر باشد که به جای تعریف مفاهیم، روی ویژگی های مهم سیستم های توزیعی تمرکز کنیم. یکی از ویژگی ها این است که تفاوت های بین کامپیوترهای مختلف و نحوه ارتباط آنها تا حد زیادی از دید کاربران مخفی می باشد . همین امر در مورد سازمان داخلی<sup>۱۴</sup> سیستم های توزیعی صدق می کند. ویژگی مهم دیگر اینکه کاربران و برنامه های کاربردی می توانند به صورتی منسجم و متحد با سیستم توزیعی ارتباط برقرار کنند، مستقل از این که این تعامل در کجا و کی صورت پذیرد.

در مجموع ، توسعه یا به مقیاس درآوردن<sup>۱۵</sup> سیستم های توزیعی باید نسبتاً به راحتی انجام شود. این ویژگی، هرچند پیامد مستقیم استقلال کامپیوترهاست ، در عین حال باعث مخفی سازی کلی نقش واقعی این کامپیوترها در سیستم می شود. حتی در صورت خرابی موقت برخی بخش های سیستم

<sup>6</sup> centralized systems

<sup>7</sup> single-processor systems

<sup>8</sup> Remote terminals

<sup>9</sup> independent computers

<sup>10</sup> coherent

<sup>11</sup> single

<sup>12</sup> autonomous components

<sup>13</sup> collaborate

<sup>14</sup> internal organization

<sup>15</sup> scale

توزیعی، این سیستم‌ها معمولاً همواره در دسترس هستند. با این وجود، کاربران و برنامه‌های کاربردی نبایستی متوجه تعمیر یا تعویض بخش‌های معیوب و یا افزودن یا عدم افزودن بخش‌های جدید برای خدمت رسانی به کاربران یا برنامه‌های کاربردی بشوند.

مطابق شکل ۱-۱، سیستم‌های توزیعی بمنظور پشتیبانی از کامپیوترها و شبکه‌های ناهمگن<sup>۱۶</sup> و در عین حال ارائه نمایی تک سیستمی، غالباً بوسیله لایه ای از نرم افزار سازماندهی می‌شوند – یعنی، بصورت منطقی بین یک لایه سطح بالاتر، شامل کاربران و برنامه‌های کاربردی و یک لایه زیرین، حاوی سیستم‌های عامل و امکانات ارتباطی قرار داده می‌شوند. به همین دلیل، این نوع سیستم توزیعی گاهی اوقات میان افزار<sup>۱۷</sup> نامیده می‌شود.

شکل ۱-۱- سیستم توزیعی که بصورت میان افزار سازماندهی شده است. لایه میان افزار به چندین ماشین گسترش یافته و برای تمامی برنامه‌های کاربردی یک رابط واحد ارائه می‌کند) داخل شکل: کامپیوتر ۱، برنامه کاربردی A، سیستم عامل محلی ۱، کامپیوتر ۲، کامپیوتر ۳، برنامه کاربردی B، سیستم عامل محلی ۲، سیستم عامل محلی ۳، کامپیوتر ۴، برنامه کاربردی C، سیستم عامل محلی ۴، لایه سیستم توزیعی (میان افزار)، شبکه).

شکل ۱-۱، چهار کامپیوتر شبکه شده و سه برنامه کاربردی را نمایش می‌دهد که برنامه کاربردی B آن در بین ماشین‌های ۲ و ۳ توزیع شده است. برای هریک از برنامه‌های کاربردی یک رابط واحد ارائه می‌شود. سیستم توزیعی امکانی را برای ارتباط مؤلفه‌های یک برنامه کاربردی منفرد با یکدیگر و همچنین برای ارتباط برنامه‌های کاربردی مختلف با یکدیگر فراهم می‌کند. در عین حال، به بهترین و منطقی‌ترین روش ممکن، تفاوت‌های بین سخت افزار و سیستم‌های عامل را از برنامه‌های کاربردی مخفی می‌کند.

## ۲-۱- اهداف

امکان ساخت سیستم‌های توزیعی لزوماً به معنای خوب بودن این ایده نیست. مثلاً هرچند فن آوری نوین امکان قرار دادن چهار فلاپی دیسک درایو روی کامپیوترهای شخصی را فراهم آورده، ولی به دلیل بی‌مورد بودن کسی حتی به فکر انجام آنهم نمی‌افتد. در این بخش، راجع به چهار هدف مهمی بحث خواهیم کرد که ساخت سیستم‌های توزیعی فقط در سایه برآورده شدن آنها موجه و معقول است: یک سیستم توزیعی باید امکان دسترسی راحت به منابع را فراهم کند؛ باید به صورتی منطقی و صحیح توزیع منابع در سرتاسر شبکه را مخفی کند؛ باید باز باشد؛ و بالاخره اینکه باید مقیاس پذیر باشد.

## ۱-۲- دسترس پذیر کردن منابع

<sup>16</sup> heterogeneous computers and networks

<sup>17</sup> middleware

هدف اصلی سیستم های توزیعی آن است که کاربران ( و برنامه های کاربردی) بتوانند به راحتی به منابع از راه دور دسترسی پیدا کرده و بصورتی کنترل شده و مؤثر، در آنها شریک<sup>۱۸</sup> شوند. این منابع ممکن است از هر نوعی باشند، اما عنوان برخی نمونه ها می توان به چاپگرهای کامپیوترها، تجهیزات ذخیره سازی، داده ها ، فایلهای، صفحات وب و شبکه ها اشاره کرد. دلایل متعددی را می توان برای لزوم اشتراک منابع ، از جمله مسائل اقتصادی ، ذکر کرد. عنوان مثال، از نظر اقتصادی به صرفه تر است که در یک دفتر کار کوچک چندین کاربر از یک چاپگر واحد استفاده کنند تا اینکه هریک مجبور به خرید و نگهداری از یک چاپگر جداگانه باشد. به همین ترتیب، از نظر اقتصادی به صرفه است که از منابع هزینه بری از قبیل ابر کامپیوترها، سیستم های ذخیره کارآ، شکل نگارها<sup>۱۹</sup> و دیگر وسائل جانبی گران قیمت به صورت اشتراکی استفاده کنیم.

علاوه، اتصال کاربران و منابع باعث تسهیل همکاری و تبادل اطلاعات می شود که از نمونه های آن می توان به موفقیت اینترنت در سایه پروتکلهای ساده آن جهت تبادل فایلهای، کارهای پستی، اسناد، صوت و شکل اشاره کرد. امروزه، اتصال به اینترنت باعث ایجاد سازمانهای مجازی<sup>۲۰</sup> متعددی شده که در آنها، گروههای انسانی که گاهی اوقات حتی چند صد هزار کیلومتر از هم فاصله دارند ، با استفاده از گروه افزار<sup>۲۱</sup> – یعنی، نرم افزاری برای ویرایش اشتراکی، کنفرانس های تلفنی و غیره - با هم کار می کنند. به همین ترتیب، ارتباطات اینترنتی با فراهم کردن امکان تجارت الکترونیک، موجب شده اند تا بتوان بدون مراجعه به فروشگاه یا حتی خروج از خانه انواع کالاها را خرید و فروش کرد.

اما همین افزایش امکان ارتباط و شراکت، نیاز روزافزون به امنیت را هم مطرح می کند. در حال حاضر، سیستم ها از محافظت کمی در برابر استراق سمع<sup>۲۲</sup> و نفوذ<sup>۲۳</sup> در ارتباط برخوردار هستند. کلمات عبور و دیگر اطلاعات حساس غالباً بصورت متن واضح<sup>۲۴</sup> (یعنی فاقد رمز) از طریق شبکه ارسال شده و یا در خدمتگزارهایی ذخیره می شوند که نمی توان اطمینان چندانی به معتبر بودن آنها داشت. در این زمینه هنوز عرصه برای بهبود باز است. عنوان مثال، امروزه خریداران می توانند صرفاً با ارائه یک شماره کارت اعتباری، حتی در صورتی هم که صاحب واقعی کارت نباشند، کالا خریداری کنند. اما در آینده، سفارش دهی به این ترتیب فقط در صورتی امکان پذیر خواهد بود که با قرار دادن کارت در کارت خوان ، دارنده کارت بصورت فیزیکی هم صاحب کارت باشد .

<sup>18</sup> share

<sup>19</sup> imagesetters

<sup>20</sup> virtual organizations

<sup>21</sup> groupware

<sup>22</sup> eavesdropping

<sup>23</sup> intrusion

<sup>24</sup> cleartext

مشکل امنیتی دیگر مربوط به ردیابی ارتباطات برای ساخت نمودار ترجیحی<sup>۲۵</sup> از کاربر مورد نظر است (Wang و گروه همکاران ۱۹۹۸). این دست ردیابی‌ها، خصوصاً اگر بدون اطلاع کاربر انجام شود، آشکارا باعث نقض حریمهای خصوصی<sup>۲۶</sup> افراد خواهد شد. مشکل دیگر این است که افزایش قابلیت ارتباط می‌تواند باعث ارتباطات ناخواسته<sup>۲۷</sup> و از جمله پست‌های الکترونیکی بیهوده‌ای شود که غالباً مزاحم<sup>۲۸</sup> نامیده می‌شوند. در این موارد، لازم است بكمک فیلترهای اطلاعاتی خاص که براساس محتوای پیام‌های ورودی اقدام به انتخاب آنها می‌کنند، از خود مراقبت کنیم.

## ۲-۱-۲- شفافیت توزیع<sup>۲۹</sup>

یکی از اهداف مهم سیستم‌های توزیعی مخفی سازی این واقعیت است که فرآیندها و منابع مربوط به آن به صورت فیزیکی در بین ماشین‌های متعدد توزیع شده‌اند. سیستم توزیعی که در نظر کاربران و برنامه‌های کاربردی خود بصورت یک سیستم کامپیوتری منفرد جلوه کند، اصطلاحاً **شفاف**<sup>۳۰</sup> نامیده می‌شود. ابتدا بباید نگاهی به انواع شفافیت‌ها در سیستم‌های توزیعی داشته باشیم. سپس به این پرسش کلی پاسخ می‌دهیم که آیا همیشه نیازمند شفافیت هستیم یا فقط گاهی اوقات به آن نیاز داریم.

### انواع شفافیت

گرچه مفهوم شفافیت را می‌توان درمورد جنبه‌های مختلف یک سیستم توزیعی بکار برد، مهمترین آنها در شکل ۱-۲ ارائه شده است.

شفافیت	شرح
دسترسی <sup>۳۱</sup>	مخفي سازی تفاوتها در ارائه داده و نحوه دسترسی به منابع
مکان <sup>۳۲</sup>	مخفي سازی محل استقرار منبع
مهاجرت <sup>۳۳</sup>	مخفي سازی احتمال جابجایی منبع به محل دیگر
مکان یابی مجدد <sup>۳۴</sup>	مخفي سازی جابجایی منبع به محل دیگر در حین استفاده از آن
کپی برداری <sup>۳۵</sup>	مخفي سازی وجود چند کپی از منبع
همروندي <sup>۳۶</sup>	مخفي سازی اشتراك منبع در بين کاربران رقيب

<sup>25</sup> preference profile

<sup>26</sup> privacy

<sup>27</sup> unwanted communication

<sup>28</sup> spam

<sup>29</sup> distribution transparency

<sup>30</sup> transparent

<sup>31</sup> Access

<sup>32</sup> Location

<sup>33</sup> Migration

<sup>34</sup> Relocation

<sup>35</sup> Replication

<sup>36</sup> Concurrency

### شکل ۱-۲- اشکال مختلف شفافیت در سیستم های توزیعی (ISO ۱۹۹۵)

**شفافیت دسترسی** به بحث در مورد مخفی سازی تفاوت های ارائه داده و نحوه دسترسی به منابع بوسیله کاربران می پردازد. در پایین ترین سطح ، هدف مخفی سازی تفاوت بین معماری ماشین هاست، اما مهمتر از آن رسیدن به توافق در زمینه نحوه ارائه داده توسط ماشین ها و سیستم عامل های مختلف است. به عنوان مثال، سیستم های کامپیوتری موجود در یک سیستم توزیعی ممکن است سیستم های عامل مختلفی را اجرا کرده و هریک از قراردادهای نامگذاری فایل<sup>۳۸</sup> مخصوص به خود استفاده کنند. تفاوت بین قراردادهای نامگذاری و همچنین نحوه دستکاری<sup>۳۹</sup> فایل ها، بایستی تماماً از کاربران و برنامه های کاربردی مخفی باشد.

گروه مهم دیگری از انواع شفافیت مربوط به محل منبع است. هدف از این **شفافیت مکان** آن است که کاربران نتوانند محل استقرار فیزیکی منبع در سیستم را شناسایی کنند. نامگذاری نقش مهمی در کسب شفافیت توزیعی دارد. شفافیت مکان بخصوص از طریق نسبت دادن اسمی منطقی<sup>۴۰</sup> به منابع - یعنی اسمی که در آنها نام محل بصورت مخفیانه کدگذاری نشده باشد - امکان‌پذیر است. عنوان نمونه، در نامگذاری URL <http://www.prenhall.com/index.html> همیشه در موقعیت فعلی خود بوده یا اخیراً به آن منتقل شده است . سیستم های توزیعی که بتوان منابع آنها را بدون تأثیرگذاری بر نحوه دسترسی به آنها انتقال داد، اصطلاحاً دارای **شفافیت مهاجرت** هستند. حالت دیگر هنگامی است که بتوان منابع را در حین دسترسی به آنها و بدون کوچکترین اطلاعی به کاربر یا برنامه کاربردی مجدداً مکان یابی کرد. در این موارد، سیستم ها اصطلاحاً از **شفافیت مکان یابی مجدد** پشتیبانی می کنند. عنوان نمونه ای از این شفافیت مکان یابی مجدد می توان به امکان ادامه استفاده کاربران متحرک از لپ تاپهای خود در حین جابجایی از یک نقطه به نقطه دیگر و بدون قطع ارتباط(حتی موقت هم) اشاره کرد.

همانطور که بعداً خواهیم دید، کپی برداری نقش مهمی در سیستم های توزیعی ایفا می کند. عنوان مثال، می توان منابع را کپی کرده و با قراردادن یکی از کپی ها در مجاورت محل مورد دسترسی به آن ، سطح کارآیی یا دسترسی را افزایش ارتقاء داد. **شفافیت کپی برداری** به مخفی سازی وجود چندین کپی از یک منبع می پردازد. برای مخفی سازی کپی ها از کاربران، تمام کپی ها باید نام واحدی داشته باشند. در نتیجه، سیستمی که از شفافیت کپی برداری پشتیبانی می کند لزوماً باید از شفافیت مکان

<sup>37</sup> Failure

<sup>38</sup> file-naming conventions

<sup>39</sup> manipulate

<sup>40</sup> logic names

یابی هم پشتیبانی کند، چون در غیراینصورت ارجاع به کپی ها در محل های مختلف غیرممکن خواهد بود.

گفتیم که یکی از اهداف مهم سیستم های توزیعی، برقراری امکان اشتراک منابع<sup>۴۱</sup> است. هرچند در بسیاری از موارد، از جمله ارتباطات، اشتراک منابع با همکاری انجام می شود، نمونه های زیادی هم می توان از اشتراک رقابتی منابع<sup>۴۲</sup> ذکر کرد. عنوان مثال، ممکن است دو کاربر مستقل فایلهای خود را روی یک خدمتگزار فایل واحد ذخیره کرده و یا به جداول واحدی در یک پایگاه داده ای مشترک<sup>۴۳</sup> دسترسی داشته باشند. در این موارد، هیچیک از کاربران نباید کوچکترین اطلاعی از واقعیت استفاده کاربر دیگر از آن منبع داشته باشند. این پدیده به نام **شفافیت همروندی** خوانده می شود. اما مهم این است که دسترسی همزنان به یک منبع مشترک، منبع را در حالت منسجم نگهادارد. انسجام<sup>۴۴</sup> از طریق مکانیزم های قفل کردن<sup>۴۵</sup> قابل ایجاد بوده و به کمک آن، کاربران دسترسی انحصاری به منبع مورد نظر خواهند داشت. مکانیزم اصلاح شده دیگر، استفاده از تراکنش ها<sup>۴۶</sup> است. اما همانطور که در فصلهای بعد خواهیم دید، پیاده سازی تراکنش ها در سیستم های توزیعی بسیار مشکل است.

لسی لمپرت تعریف دیگری برای سیستم های توزیعی ارائه می دهد: "هنگامی خواهید دانست یک سیستم توزیعی دارید که شکست یک کامپیوتر که نشنیده اید، ادامه کار شما را متوقف کند". این تعریف دقیقاً روی یکی دیگر از مسائل مهم در طراحی سیستم های توزیعی، یعنی مبحث خرابی ها، انگشت می گذارد. ایجاد **شفافیت خرابی** در سیستم های توزیعی به این معناست که کاربر متوجه خرابی و عملکرد نادرست یک منبع نشده (حتی شاید کوچکترین گزارشی هم از آن دریافت نکند) و سپس سیستم اقدام به ترمیم آن خرابی کند. مخفی سازی خرابی ها یکی از مشکل ترین مسائل در سیستم های توزیعی بوده و همانطور که در فصل ۸ خواهیم گفت، برخی فرضیات بظاهر واقع بینانه آنرا حتی غیرممکن هم می کند. مشکل اصلی در مخفی سازی خرابی ها ناشی از عدم توانایی در تمایز بین منبع مرده<sup>۴۷</sup> از منبع کند<sup>۴۸</sup> است. مثلاً در بعضی موارد، پس از چندین تماس با یک خدمتگزار وب مشغول، جستجوگر به سررسید زمانی می رسد و گزارش می دهد که صفحه وب مورد نظر غیرقابل دسترسی است. در این حالت کاربر نباید نتیجه گیری کند که خدمتگزار واقعاً از کارافتاده است.

### درجه شفافیت

<sup>41</sup> sharing of resources

<sup>42</sup> competitive sharing of resources

<sup>43</sup> shared database

<sup>44</sup> Consistency

<sup>45</sup> locking mechanisms

<sup>46</sup> transaction

<sup>47</sup> dead resource

<sup>48</sup> slow resource

هرچند شفافیت توزیعی معمولاً برای تمامی سیستم‌های توزیعی مناسب تصور می‌شود، در برخی موارد مخفی سازی کامل تمامی جنبه‌های توزیع از کاربران چندان هم قابل قبول نیست. بعنوان نمونه، مثلاً تمایل دارید که روزنامه الکترونیکی مورد علاقه تان همیشه پیش از ساعت هفت صبح به وقت محلی در صندوق پستی تان قرار بگیرد، در حالیکه ممکن است در آن ساعت در جای دیگری از دنیا و در محدوده زمانی دیگری زندگی کنید. بنابراین ممکن است با دیدن روزنامه "به اصطلاح صحبتان"<sup>49</sup> خیلی تعجب کنید. همچنین نمی‌توان انتظار داشت که سیستم توزیعی ناحیه گسترده‌ای که فرآیندی را در سانفرانسیسکو به فرآیندی در آمستردام متصل می‌کند، بتواند طرف کمتر از ۳۵ میلی ثانیه پیامی را از یک فرآیند به فرآیند دیگر ارسال و قانون طبیعت را مخفی کند. در عمل، اینکار با استفاده از یک شبکه کامپیوترا صورت می‌گیرد و چندصد میلی ثانیه طول می‌کشد. انتقال سیگنال علاوه بر وابستگی به سرعت نور، با ظرفیت‌های پردازشی سوییچ‌های میانی هم مرتبط است.

همچنین نوعی توازن<sup>۵۰</sup> بین درجه بالای شفافیت و کارایی سیستم وجود دارد. بعنوان مثال، بسیاری از برنامه‌های کاربردی اینترنت ممکن است بارها برای اتصال به یک خدمتگزار تلاش کرده و درصورت عدم موفقیت از اینکار صرفنظر کنند. در نتیجه، تلاش برای مخفی سازی خرابی وقت خدمتگزار پیش از آزمودن خدمتگزار دیگر ممکن است در کل باعث کاهش سرعت سیستم شود. در این موارد، بهتر است کاربر زودتر منصرف شود و یا حداقل از تلاشهای مکرر برای ایجاد اتصال صرفنظر کند.

در نمونه دیگر مثلاً باید انسجام همیشگی کپی‌های مختلف که در قاره‌های متفاوت واقع شده اند را تضمین کنیم. به بیان دیگر، درصورت تغییر یکی از کپی‌ها، این تغییر باقیتی پیش از هر عمل دیگری به تمام کپی‌ها انتشار پیدا کند. واضح است که تکمیل فقط یک عمل به روزآوری ممکن است چندین ثانیه طول بکشد، و این چیزی است که نمی‌توان از کاربران مخفی کرد.

و بالاخره اینکه در برخی موارد، نمی‌توان بطور قطع ادعا کرد که مخفی سازی شفافیت ایده درستی است. از آنجایی که سیستم‌های توزیعی در حال گسترش به دستگاه‌های قابل حملی هستند که اهمیت آگاهی از مکان و بافت<sup>۵۱</sup> در آنها بیش از پیش در حال افزایش است، شاید بهتر باشد که بجای مخفی سازی شفافیت حتی آنرا عملًا هم در معرض دید قرار دهیم. اهمیت این بی‌حفظی در برابر توزیع در سیستم‌های توزیعی همه جا حاضر<sup>۵۲</sup> و نهفته<sup>۵۳</sup> در ادامه همین فصل بیشترآشکار خواهد شد. بعنوان یک نمونه ساده، در نظر بگیرید که کارمندی بخواهد فایلی را از نوت بوک خود چاپ کند. در اینصورت بهتر است که نسخه چاپی را به یک چاپگر نزدیک مشغول ارسال کنیم تا به چاپگر بیکاری در دفتر اصلی شرکت در کشور دیگر.

<sup>49</sup> trade-off

<sup>50</sup> Context

<sup>51</sup> ubiquitous

<sup>52</sup> embedded

استدلال های دیگری را هم می توان علیه شفافیت توزیعی مطرح کرد. با این فرض که دستیابی به شفافیت توزیعی کامل تقریباً غیرممکن است، باید از خود بپرسیم که آیا حتی تظاهر به برخورداری از این ویژگی عاقلانه هست یا خیر. شاید بسیار بهتر باشد توزیع را چنان آشکار کنیم که کاربر و سازنده برنامه کاربردی هیچگاه حتی به فکر شفافیت هم نیفتند . در نتیجه، کاربران بهتر رفتار (شاید غیرمنتظره) سیستم توزیعی را درک کرده و بنابراین آمادگی بسیار بیشتری برای برخورد با این رفتار پیدا خواهند کرد.

بنابراین، هرچند هدف قرار دادن شفافیت توزیعی در طراحی و پیاده سازی سیستم های توزیعی ممکن است پسندیده باشد، اما باید به همراه آن مسائل دیگری از قبیل کارآیی و جامعیت<sup>۵۳</sup> هم لحاظ شود. هزینه کسب شفافیت کامل ممکن است واقعاً کمرشکن باشد.

### ۳-۱-۲-۳- بازبودن<sup>۵۴</sup>

یکی دیگر از اهداف مهم سیستم های توزیعی، بازبودن است . سیستم توزیعی باز<sup>۵۵</sup> سیستمی است که براساس قوانین استاندارد تشريح کننده معنا و نحو<sup>۵۶</sup> خدمات ، اقدام به ارائه آنها می کند. عنوان مثال، در شبکه های کامپیوترا، قوانین استانداردی بر قالب، محتوا و معنای پیامهای ارسالی و دریافتی حاکم است. این دست قوانین در قالب پروتکل مدون می شوند. در سیستم های توزیعی، سرویس ها غالباً از طریق رابط ها<sup>۵۷</sup> تخصیص یافته و غالباً توسط زبان تعریف رابط(IDL)<sup>۵۸</sup> تشريح می شوند. تعاریف رابط نوشته شده در IDL ، تقریباً همیشه فقط درمورد نحو خدمات صحبت می کنند . به بیان دیگر، صرفاً اسمی توابعی را تعیین می کند که به همراه انواع پارامترها، مقادیر بازگشت<sup>۵۹</sup> ، استثنایات احتمالی و غیره موجود هستند. بخش مشکل کار تخصیص وظیفه دقیق این سرویس ها، یعنی معنای رابط ها است. در عمل، این دست ویژگی ها همیشه به صورتی غیر متعارف، یعنی بوسیله زبان طبیعی ارائه می شوند.

در صورتی که رابط به نحو مناسبی تعریف شود، اجازه می دهد یک فرآیند دلخواه که این رابط را دارد با فرآیند دیگری که این رابط را ارائه می دهد صحبت نماید. همچنین باعث می شود تا دو طرف مستقل، پیاده سازی های کاملاً متفاوتی را از آن رابط ایجاد کرده و در نتیجه، دو سیستم توزیعی جداگانه، اما با عملکرد یکسان، بوجود می آید . اما این ویژگی ها باید کامل<sup>۶۰</sup> و خنثی<sup>۶۱</sup> باشند. کامل بودن یعنی، تمام آنچه برای پیاده سازی لازم است واقعاً تعریف شده باشد. با این وجود، بسیاری از تعاریف رابط به هیچ

<sup>53</sup> comprehensiveness

<sup>54</sup> Openness

<sup>55</sup> open distributed system

<sup>56</sup> semantics and syntax

<sup>57</sup> interfaces

<sup>58</sup> interface definition language

<sup>59</sup> return values

<sup>60</sup> Complete

<sup>61</sup> Neutral

وجه کامل نبوده و سازنده آنها باید جزئیات مربوط به پیاده سازی را به آنها اضافه کند. به همین اندازه خنثی بودن اهمیت دارد به این معنی که نحوه و شکل پیاده سازی ویژگی ها تجویز نشود. کامل بودن و خنثی بودن باعث کسب قابلیت کار با هم<sup>62</sup> و قابلیت جابجایی<sup>63</sup> می شود (Blair و Stefani ۱۹۹۸).

**قابلیت کار با هم** بیان می کند که اشکال مختلف پیاده سازی سیستم ها یا مؤلفه های سازنده گان مختلف تا چه حد می توانند صرفاً با تکیه بر سرویس های یکدیگر که بوسیله یک استاندارد مشترک تعریف شده است، همزیستی و با هم کار کنند. **قابلیت جابجایی** بیان می دارد که برنامه کاربردی ایجاد شده برای سیستم توزیعی مفروض A تا چه حد، بدون نیاز به تغییر و اصلاح، روی سیستم توزیعی دیگر B که همان رابط های A را پیاده سازی کرده است، قابل اجراست.

یکی دیگر از اهداف مهم سیستم های توزیعی باز آن است که پیکره بندی سیستم توسط مؤلفه های مختلف (شاید از سازنده گان مختلف) براحتی امکان پذیر باشد. همچنین، افزودن مؤلفه های جدید یا تعویض مؤلفه های فعلی نباید هیچ تأثیری بر دیگر مؤلفه ها داشته باشد. به بیان دیگر، یک سیستم توزیعی باز باید **قابل گسترش**<sup>64</sup> هم باشد. بعنوان مثال، در یک سیستم قابل گسترش ، افزودن بخشهایی که روی یک سیستم عامل متفاوت اجرا می شوند یا حتی تعویض کل سیستم فایل باید تقریباً به راحتی انجام شود. اما باید اعتراف کرد که کسب انعطاف پذیری در حرف بسیار آسانتر از عمل است.

### تفکیک سیاست از مکانیزم

برای کسب انعطاف پذیری<sup>65</sup> در سیستم های توزیعی باز ، سیستم بایستی لزوماً بصورت مجموعه ای از مؤلفه های براحتی قابل انطباق یا قابل تعویض و نسبتاً کوچک سازماندهی شود. به این معنا که، بایستی تعاریفی را هم برای رابط های بالاترین سطح – یعنی رابط هایی که بوسیله کاربران و برنامه های کاربردی قابل روئیت هستند- و هم برای رابط های بخشهای داخلی سیستم ایجاد و نحوه ارتباط این بخشها را توصیف کرد. این روش تقریباً جدید است. بسیاری از سیستم های قدیمی تر و حتی تقریباً جدید ، براساس شیوه یکپارچه ای سازماندهی شده اند که در آن، مؤلفه ها فقط به صورت منطقی تفکیک شده ولی به صورت یک برنامه واحد و عظیم پیاده سازی شده اند . این روش، تعویض یا انطباق مؤلفه ها بدون تأثیرگذاری بر کل سیستم را مشکل می نماید. بنابراین ، سیستم های یکپارچه<sup>66</sup> ، به جای باز بودن، تمایل به بسته بودن دارند .

<sup>62</sup> Interoperability

<sup>63</sup> Portability

<sup>64</sup> extensible

<sup>65</sup> flexibility

<sup>66</sup> monolithic approach

لزوم تغییر سیستم‌های توزیعی غالباً ناشی از مؤلفه‌ای است که سیاست بهینه‌ای را برای یک کاربر یا برنامه کاربردی ایجاد نمی‌کند. بعنوان مثال، ذخیره نهان<sup>۶۷</sup> در شبکه جهانی وب را در نظر بگیرید. جستجوگرها عموماً به کاربرها امکان می‌دهند تا با تعیین ابعاد حافظه نهان و لزوم یا عدم لزوم چک کردن دائمی سند ذخیره نهان شده از نظر انسجام ، و یا چک کردن انسجام فقط برای یک مرتبه در طول جلسه ، سیاست ذخیره نهان<sup>۶۸</sup> خود را منطبق کنند. با این وجود ، کاربر نمی‌تواند تغییری در دیگر پارامترهای ذخیره نهان، از قبیل مدت زمان ماندگاری سند در حافظه نهان یا تعیین سندی که باید با پرشدن حافظه نهان حذف شود ، ایجاد کند. همچنین، نمی‌توان براساس محتوای سند راجع به مخفی سازی آن تصمیم گیری کرد. بعنوان مثال، یک کاربر ممکن است با اطلاع از اینکه جداول زمانی راه آهن به ندرت تغییر می‌کنند ، این اطلاعات ، و نه اطلاعات مربوط به شرایط فعلی ترافیک در بزرگراهها، را در حافظه نهان قرار دهد .

در اینجا لازم است که بین مکانیزم و سیاست تفاوت قائل شویم. بعنوان مثال، در مورد ذخیره نهان شبکه جهانی وب، جستجوگر بایستی امکاناتی را صرفاً جهت ذخیره سازی اسناد فراهم کرده و در عین حال ، به کاربران اجازه دهد تا راجع به اسناد قابل ذخیره شده و مدت زمان ذخیره تصمیم گیری کند. در عمل، اینکار از طریق ارائه مجموعه کاملی از پارامترهای قابل تنظیم برای کاربر (به صورتی پویا) پیاده می‌شود. حتی بهتر است که کاربر بتواند سیاست خود را به شکل یک مؤلفه قابل اتصال به جستجوگر پیاده کند. البته، مؤلفه مذکور بایستی یک رابط قابل فهم برای جستجوگر را داشته باشد تا جستجوگر بتواند روال‌های این رابط را فراخوانی کند.

#### **۴-۲-۱- مقیاس پذیری<sup>۶۹</sup>**

امروزه اتصال جهانی از طریق اینترنت، مانند امکان ارسال یک کارت پستال برای هرگوشه ای از جهان، تبدیل به امری عادی شده است . به همین دلیل ، مقیاس پذیری یکی از مهمترین اهداف طراحی برای سازندگان سیستم‌های توزیعی محسوب می‌شود.

مقیاس پذیری یک سیستم را می‌توان حداقل در سه بعد مختلف اندازه گیری نمود (Neuman ۱۹۹۴). اولاً ، یک سیستم می‌تواند با توجه به اندازه خود مقیاس پذیر باشد، به این معنا که بتوان به راحتی کاربران و منابع دیگری را به سیستم اضافه نمود. ثانياً، یک سیستم مقیاس پذیر جغرافیایی سیستمی است که ممکن است کاربران و منابع آن در فاصله‌های دوری از هم قرار گرفته باشند. ثالثاً ، یک سیستم ممکن است از نظر مدیریت اجرایی مقیاس پذیر باشد، به این معنا که حتی اگر سازمان هایی با مدیریت اجرایی مستقل را هم بهم پیوند دهد، باز به راحتی قابل مدیریت باشد. متأسفانه، اغلب

<sup>67</sup> caching

<sup>68</sup> caching policy

<sup>69</sup> scalability

سیستم هایی که از یک یا چند بعد مقیاس پذیر هستند ، با افزایش مقیاس پذیری سیستم، تا حدودی با افت عملکرد مواجه می شوند .

### مشکلات مقیاس پذیری

پیش از ایجاد امکان مقیاس پذیری در یک سیستم باید انواع مختلفی از مشکلات را حل و فصل نمود. ابتدا بباید نگاهی به مسئله مقیاس پذیری بر اساس اندازه داشته باشیم. پشتیبانی از تعداد بیشتری از کاربران یا منابع غالباً با محدودیتهای سرویس ها، داده ها و الگوریتمهای مت مرکز مواجه می شویم (مراجعةه کنید به شکل ۱-۳). بعنوان مثال، بسیاری از سرویس ها از این نظر مت مرکز هستند که بوسیله فقط یک خدمتگزار روی یکی از ماشین های سیستم توزیعی پیاده می شوند. مشکل این دست طرحها مشخص است: با افزایش تعداد کاربران و برنامه های کاربردی، ممکن است خدمتگزار تبدیل به یک گلوگاه شود. حتی در صورت نامحدود بودن ظرفیت ذخیره سازی و پردازش، برقراری ارتباط با این دست خدمتگزارها مانعی در برابر رشد خواهد بود.

متأسفانه، گاهی اوقات مجبور هستیم که فقط از یک خدمتگزار استفاده کنیم . تصور کنید که برای مدیریت اطلاعات بسیار محرومراهی از قبیل گزارشات پژوهشی، حسابهای بانکی و غیره یک سرویس در اختیار داشته باشیم. در این موارد، یکی از بهترین گزینه ها پیاده سازی آن سرویس با استفاده از یک سرویس جداگانه و در یک اتاق جداگانه بسیار حافظت شده و این از بخشهای دیگر سیستم توزیعی از طریق مؤلفه های شبکه ای خاص باشد. کپی کردن خدمتگزار در محلهای مختلف از آنجایی که باعث کاهش سطح امنیتی سرویس می شود امکان پذیر نمی باشد.

مثال	مفهوم
فقط یک سیستم برای تمام کاربران	سرویس های مت مرکز
فقط یک دفترچه تلفن واحد آن لاین	داده های مت مرکز
مسیریابی براساس اطلاعات کامل شبکه	الگوریتم های مت مرکز

شکل ۱-۳- نمونه هایی از محدودیت های مقیاس پذیری

داده های مت مرکز درست به اندازه سرویس های مت مرکز نامناسب هستند. چگونه باید شماره تلفن و آدرس پنجاه میلیون نفر را پی گیری کرد؟ فرض کنید هر رکورد داده ای ۵۰ کاراکتر باشد. بنابراین ، یک بخش ۲/۵ گیگابایتی دیسک فضای ذخیره کافی برای اینکار ایجاد می کند. اما در اینجا هم داشتن فقط یک پایگاه داده ای باعث اشغال شدن تمام خطوط ارتباطی داخلی و خارجی خواهد شد. همچنین تصور کنید اگر سیستم نام دامنه<sup>۷۰</sup> (DNS) بعنوان یک جدول منفرد پیاده شود، چه عملکردی خواهد داشت ؟ DNS اطلاعات مربوط به میلیون ها کامپیوتر در سرتاسر جهان را نگه داشته و یکی از

<sup>۷۰</sup> Domain Name System

سرویس اصلی برای مکان یابی خدمتگزارهای وب محسوب می شود. اما اگر قرار بود که هریک از درخواست های تجزیه<sup>۷۱</sup> URL را به یک و تنها خدمتگزار DNS ارسال کرد ، هیچکس تمایلی به استفاده از وب نمی داشت .

نهایتاً اینکه الگوریتمهای متمرکز هم چندان مناسب نیستند. در یک سیستم توزیعی بزرگ، تعداد زیادی پیام باید روی تعداد زیادی خط مسیردهی شوند. از نقطه نظر تئوری، روش بهینه برای انجام اینکار، جمع آوری اطلاعات کامل درباره بار تمامی ماشین ها و خطوط و سپس اجرای الگوریتمی برای محاسبه مسیرهای بهینه<sup>۷۲</sup> است. سپس می توان جهت ارتقاء مسیردهی، این اطلاعات را در اطراف سیستم پراکنده کرد.

مشکل در آنجاست که جمع آوری و انتقال تمامی اطلاعات ورودی و خروجی هم ایده مناسبی نیست، چون این پیامها بخشی از اضافه بار شبکه را ایجاد می کنند. در واقع، از هر الگوریتمی که براساس جمع آوری اطلاعات از تمامی سایتها عمل می کند و اطلاعات را جهت پردازش به یک ماشین مشخص ارسال کرده و سپس نتایجی را توزیع می کند اجتناب شود . برای این منظور فقط باید از الگوریتم های غیرمتمرکز استفاده شود . این الگوریتم ها معمولاً دارای ویژگی های زیر بوده که وجه تمایز آنها از الگوریتم های متمرکز محسوب می شود:

۱. هیچیک از ماشین ها اطلاعات کاملی راجع به وضعیت سیستم در اختیار ندارد.
۲. ماشین ها فقط براساس اطلاعات محلی تصمیم گیری می کنند.
۳. خرابی یک ماشین باعث خرابی الگوریتم نمی شود.
۴. هیچ فرض ضمنی راجع به وجود نوعی ساعت جهانی<sup>۷۳</sup> وجود ندارد.

از آنچه تا به اینجا گفته شد سه حالت اول به راحتی قابل درک و موجه است . مورد آخر هم گرچه خیلی واضح نیست اما مهم است. از آنجایی که همگام سازی دقیق تمام ساعتها غیرممکن است، بنابراین هر الگوریتمی که با عبارتی مشابه " دقیقاً در ۱۲:۰۰:۰۰ تمام ماشین ها باید اندازه صفحه خروجی خود را ثبت کنند" آغاز شود، با شکست مواجه خواهد شد. در الگوریتم ها بایستی عدم همگامی دقیق ساعتها مدنظر قرار گیرد. هرقدر که سیستم بزرگ تر باشد، عدم قطعیت هم بیشتر افزایش خواهد یافت. در یک LAN منفرد، شاید بتوان با صرف تلاش کافی همه ساعت ها را تا چند میلی ثانیه همگام کرد ، اما انجام اینکار در سطح کشوری یا بین المللی آرزویی دست نیافتندی است.

مقیاس پذیری جغرافیایی<sup>۷۴</sup> هم مشکلات مخصوص به خود را دارد. یکی از مهمترین دلایل بروز اشکال در مقیاس پذیری سیستم های توزیعی موجود که برای شبکه های محلی طراحی شده اند این است که

<sup>71</sup> resolve

<sup>72</sup> optimal routes

<sup>73</sup> global clock

<sup>74</sup> geographical scalability

براساس ارتباط همگام<sup>۷۵</sup> استوار هستند. در این شکل از ارتباط، طرف درخواست کننده سرویس که معمولاً به نام مشتری<sup>۷۶</sup> خوانده می‌شود، تا زمان بازگشت پاسخ مسدود باقی می‌ماند. این روش معمولاً برای LAN‌های مناسب است که در آنها ارتباطات بین دو ماشین بطور معمول حداکثر چند صد میلی ثانیه طول می‌کشد. اما در سیستم‌های حوزه گسترده، ارتباطات میان فرآیندی ممکن است صدها میلی ثانیه یعنی هزار برابر بیشتر طول بکشد. ساخت برنامه‌های کاربردی تعاملی<sup>۷۷</sup> با استفاده از ارتباط همگام در سیستم‌های حوزه گسترده مستلزم صرف دقت زیادی است.

مشکل دیگر در کسب مقیاس پذیری جغرافیایی این است که ارتباطات در شبکه‌های حوزه گسترده ذاتاً نامطمئن و تقریباً همیشه نقطه-به-نقطه<sup>۷۸</sup> است. بالعکس، شبکه‌های منطقه محلی معمولاً امکانات ارتباطی بسیار مطمئن براساس پخش<sup>۷۹</sup> ایجاد می‌کنند و همین امر باعث سادگی زیاد در ایجاد سیستم‌های توزیعی می‌شود. بعنوان مثال، مسئله یافتن یک سرویس را در نظر بگیرید. در سیستم‌های محلی، فرآیند می‌تواند با استفاده از پخش پیام به تمامی ماشین‌ها از آنها درباره سرویس مورد نیاز خود پرسش نماید. فقط آنسته از ماشین‌هایی که سرویس را در اختیار داشته باشند پاسخ داده و هریک آدرس شبکه‌ای خود را در پیام پاسخ قرار می‌دهد. استفاده از چنین طرح‌های مکان یابی در سیستم‌های حوزه گسترده ای از قبیل اینترنت غیرممکن است. در عوض، سرویس‌های مکان یابی خاص باید طراحی شوند که مقیاس پذیر در سطح جهانی بوده و قادر به سرویس دهی به یک میلیارد کاربر باشند. در فصل ۵ مجدداً راجع به این سرویس‌ها بحث خواهیم کرد.

مقیاس پذیری جغرافیایی ارتباط زیادی با مشکلات راه حل‌های مرکزی دارد که مانع از مقیاس پذیری اندازه<sup>۸۰</sup> می‌شود. اگر یک سیستم با مؤلفه‌های مرکزی متعدد در اختیار داشته باشیم، مقیاس پذیری جغرافیایی به دلیل مشکلات کارآیی و قابلیت اطمینان ناشی از ارتباط حوزه گسترده دچار محدودیت خواهد بود. بعلاوه، مؤلفه‌های مرکزی باعث اتلاف منابع شبکه می‌شوند. تصور کنید که برای تمام کشور فقط از یک خدمتگزار پست الکترونیکی استفاده شود. در اینصورت، پست الکترونیکی که برای همسایه تان ارسال می‌کنید باید ابتدا به خدمتگزار پستی مرکزی ارسال شود که ممکن است صدها کیلومتر از شما فاصله داشته باشد. مشخص است که این روش به هیچ وجه مناسب و مفید نیست.

نهایتاً، یکی از سوالات مشکل و در برخی موارد باز درباره نحوه مقیاس پذیری یک سیستم توزیعی در بین چندین دامنه مدیریتی مستقل است. یکی از مهمترین مشکلات در این رابطه، تضاد سیاست‌ها در رابطه با به کارگیری (و پرداخت) و مدیریت و ایمنی منابع است.

<sup>75</sup> synchronous communication

<sup>76</sup> client

<sup>77</sup> interactive applications

<sup>78</sup> point-to-point

<sup>79</sup> Broadcast

<sup>80</sup> size scalability

بعنوان مثال، بسیاری از مؤلفه های یک سیستم توزیعی که در داخل یک دامنه مستقر هستند، غالباً مورد اعتماد کاربرانی است که در داخل همان دامنه عمل می کنند . در این موارد، مدیریت اجرایی سیستم ممکن است برنامه های کاربردی آزموده و تأیید شده ای در اختیار داشته و یا برای تضمین عدم دستکاری در این مؤلفه ها اقدامات خاصی اتخاذ کند. بطور خلاصه، کاربران به مدیران اجرایی سیستم خود اعتماد می کنند. با این وجود، این اعتماد طبیعتاً در سرتاسر مرازهای دامنه توسعه نمی یابد.

چنانچه یک سیستم توزیعی به دامنه دیگر گسترش یابد، بایستی دو نوع اقدام امنیتی صورت پذیرد. در درجه اول، سیستم توزیعی باید از خود در برابر حملات ناخواسته از دامنه جدید محافظت کند. عنوان مثال، کاربران دامنه جدید ممکن است به سیستم فایل در دامنه اصلی فقط دسترسی خواندن داشته باشند. به همین ترتیب، امکاناتی از قبیل شکل نگار های گران قیمت یا کامپیوترهای پریازده ممکن است برای کاربران خارج از دامنه قابل دستیابی نباشد. ثانیاً، دامنه جدید بایستی از خود در برابر حملات ناخواسته از جانب سیستم توزیعی محافظت کند. عنوان یک نمونه می توان به دانلود کردن برنامه هایی از قبیل ریزبرنامه ها<sup>۸۱</sup> در جستجوگرهای وب اشاره کرد . اساساً، دامنه جدید نمی دارد که باید چه نوع رفتاری را از این کدهای خارجی انتظار داشته باشد و بنابراین ممکن است تصمیم به محدود کردن شدید حقوق دسترسی<sup>۸۲</sup> این کدها بگیرد . همانطور که در فصل ۹ خواهیم دید، نحوه اعمال این محدودیتها خود یکی از مشکلات است .

### تکنیکهای مقیاس پذیری

بحث در مورد برخی از مشکلات مقیاس پذیری ما را به مسئله نحوه حل و فصل این مشکلات سوق می دهد. در اغلب موارد، مشکلات مقیاس پذیری در سیستم های توزیعی به صورت مشکلات کارآیی ناشی از محدودیت ظرفیت خدمتگزارها و شبکه ها جلوه می کند . در حال حاضر، اساساً فقط سه تکنیک در زمینه مقیاس پذیری وجود دارد: مخفی سازی تأخیرهای ارتباطات<sup>۸۳</sup>، توزیع و کپی برداری<sup>۸۴</sup> [مراجهعه شود به (Neuman ۱۹۹۴)].

مخفی سازی تأخیرهای ارتباطی تأثیر مهمی در کسب مقیاس پذیری جغرافیایی دارد. ایده اصلی، اجتناب حداقلی از انتظار برای دریافت پاسخ به درخواستهای سرویس از راه دور( و احتمالاً در فاصله دور) است. عنوان مثال، وقتی سرویس واقع در ماشین از راه دوری درخواست می شود، می توان بجای انتظار برای دریافت پاسخ از جانب خدمتگزار، کار مفید دیگری در طرف درخواست کننده انجام داد . اساساً، این به معنای ساخت برنامه کاربردی درخواست کننده به صورتی است که فقط از ارتباط ناهمگام<sup>۸۴</sup> استفاده کند. با ورود پاسخ، برنامه کاربردی متوقف شده و برای تکمیل درخواست قبلاً صادر

<sup>81</sup> applets

<sup>82</sup> access rights

<sup>83</sup> communication latencies

<sup>84</sup> asynchronous communication

شده، عمل کننده<sup>۸۵</sup> خاصی فراخوانی می شود. از ارتباطات ناهمگام غالباً می توان در سیستم های پردازش دسته ای<sup>۸۶</sup> و برنامه های کاربردی موازی<sup>۸۷</sup> استفاده کرد که در آنها ، وظیفه های کمابیش مستقل را می توان در حالی برای اجرا زمانبندی کرد که کار دیگری منتظر تکمیل ارتباط است . روش دیگر ، آغاز یک نخ کنترلی<sup>۸۸</sup> جدید جهت انجام درخواست است . هرچند این نخ انتظار برای پاسخ را مسدود می کند، نخ های دیگر فرآیند قادر به ادامه کار هستند.

با این وجود، بسیاری از برنامه های کاربردی نمی توانند از ارتباطات ناهمگام بصورتی مفید و کارآمد استفاده کنند. بعنوان مثال، در برنامه های کاربردی تعاملی<sup>۸۹</sup> وقتی کاربر درخواستی را ارسال می کند، معمولاً مجبور است که منتظر پاسخ باقی بماند. در این موارد، راه حل بهتر کاهش کلی ارتباط، مثلاً، از طریق انتقال بخشی از محاسبه که معمولاً در خدمتگزار انجام می شود به فرآیند مشتری درخواست کننده سرویس است. یکی از نمونه های اثربخشی این روش ، دسترسی به پایگاه های داده با استفاده از فرم ها است. پرکردن فرم از طریق ارسال پیامی جداگانه برای هریک از فیلدها و انتظار برای دریافت تصدیق از جانب خدمتگزار، مطابق شکل ۱-۴ (الف)، انجام می شود. بعنوان مثال، خدمتگزار ممکن است پیش از پذیرش ورودی خطاهای نحوی آنرا چک کند. بر اساس شکل ۱-۴ (ب)، راه حل بسیار بهتر انتقال کد مربوط به پرکردن فرم ، و احتمالاً چک کردن ورودی ها، برای مشتری و بازگرداندن فرم پرشده توسط مشتری است. این روش انتقال کد امروزه بطور گسترده توسط وب به صورت ریزبرنامه های جاوا<sup>۹۰</sup> و جاوا اسکریپت<sup>۹۱</sup> پشتیبانی می شود.

**شکل ۱-۴- تفاوت بین بررسی فرمها توسط الف) خدمتگزار و ب) مشتری در حین تکمیل آنها( داخل شکل: الف) مشتری، نام کوچک، نام خانوادگی، پست الکترونیک، خدمتگزار، چک کردن فرم، پردازش فرم، ب) مشتری، نام کوچک، نام خانوادگی، پست الکترونیک، چک کردن فرم ، خدمتگزار، پردازش فرم).**

یکی دیگر از تکنیکهای مهم مقیاس پذیری، **توزیع**<sup>۹۲</sup> است. توزیع به معنای انتخاب یک مؤلفه، تجزیه آن به قطعات کوچکتر و سپس پخش این قطعات در بین سیستم هاست. یکی از بهترین نمونه های توزیع ، سیستم نام دامنه (DNS) است. مطابق شکل ۱-۵، فضای نام DNS بصورت سلسله مراتبی به صورت سه دامنه<sup>۹۳</sup> سازمان داده شده و هر دامنه به مناطق<sup>۹۴</sup> نا هم پوشان<sup>۹۵</sup> تقسیم می شود. اسامی در

<sup>85</sup> Handler

<sup>86</sup> batch-processing systems

<sup>87</sup> parallel application

<sup>88</sup> thread of control

<sup>89</sup> interactive applications

<sup>90</sup> Java applets

<sup>91</sup> Javascript

<sup>92</sup> distribution

<sup>93</sup> domain

<sup>94</sup> Zones

<sup>95</sup> nonoverlapping

هریک از مناطق بوسیله یک خدمتگزار نام واحد اداره می شود. بدون وارد شدن به جزئیات می توان تصور کرد که نام هر مسیر در واقع نام میزبانی در اینترنت خواهد بود و بنابراین با آدرس شبکه ای آن میزبان مرتبط است. اساساً، تجزیه نام به معنای بازگشت آدرس شبکه ای میزبان مربوطه است. عنوان مثال، نام *nl.vu.cs.flits* را در نظر بگیرید . برای تجزیه، ابتدا این نام به خدمتگزار منطقه Z1 داده شده (مراجعه شود به شکل ۱-۵) و آنهم آدرس خدمتگزار منطقه Z2 را باز می گرداند که مابقی نام *vu.cs.flits* به آن داده می شود. خدمتگزار مربوط به Z2 آدرس خدمتگزار مربوط به منطقه Z3 یعنی را باز می گرداند که آنهم با کاربروی بخش پایانی نام، آدرس میزبان مربوطه را بازمی گرداند.

**شکل ۱-۵ - نمونه ای از تقسیم فضای نام DNS به چندین ناحیه (داخل شکل: عمومی، کشورها).**

این مثال نمونه بارزی از توزیع سرویس نامگذاری<sup>۹۶</sup>، که بوسیله DNS ایجاد می شود، در بین چندین ماشین و بنابراین جلوگیری از اشتغال فقط یک خدمتگزار با تمامی درخواستهای مربوط به تجزیه نام<sup>۹۷</sup> است.

بعنوان مثال دیگر، شبکه جهانی وب را در نظر بگیرید. برای اکثر کاربران، وب یک سیستم اطلاعاتی عظیم براساس سند است که هریک از اسناد آن دارای نام منحصر به فردی به شکل URL است. از نظر مفهومی، حتی ممکن است گاهی تصور شود که فقط یک خدمتگزار وجود دارد. در حالیکه وب از نظر فیزیکی در بین تعداد زیادی از خدمتگزارها توزیع شده که هریک روی تعدادی از اسناد وب کار می کند. نام خدمتگزاری که روی سند کار می کند در URL آن سند کد گذاری می شود. دقیقاً به خاطر همین ویژگی توزیع اسناد است که وب تا به حال قادر به مقیاس پذیری با ابعاد فعلی خود بوده است. با این فرض که مشکلات مقیاس پذیری غالباً به شکل افت کارآیی ظاهر می شود، معمولاً بهتر است که مؤلفه ها را واقعاً در داخل سیستم توزیعی کپی برداری<sup>۹۸</sup> کنیم. این کپی برداری علاوه بر افزایش قابلیت دسترسی ، به تعادل بار در بین مؤلفه ها کمک کرده و در نتیجه باعث افزایش کارآیی هم خواهد شد. بعلاوه، در سیستم های دارای پراکندگی جغرافیایی زیاد ، در اختیار داشتن یک کپی دم دستی باعث مخفی سازی بخش عمده ای از مشکلات تأخیر در ارتباطات می شود که قبلاً مورد بحث قرار گرفت.

هر چند به کار گیری حافظه نهان<sup>۹۹</sup> شکل خاصی از کپی برداری است، تفاوت های جزئی بین آنها وجود دارد. به کار گیری حافظه نهان هم مانند کپی برداری، باعث ایجاد یک کپی از منبع معمولاً در نزدیکی مشتری می شود که به آن منبع دسترسی دارد. در حالیکه برخلاف کپی برداری تصمیمی است که

<sup>96</sup> naming service

<sup>97</sup> name resolution

<sup>98</sup> replicate

<sup>99</sup> caching

بوسیله مشتری منبع گرفته می شود و نه بوسیله مالک منبع. بعلاوه، حافظه نهان در صورت اعلام نیاز به کار گرفته می شود، در حالیکه کپی برداری طرحی از پیش برنامه ریزی شده است. اما به کار گیری حافظه نهان و کپی برداری یک نقطه ضعف عمدۀ دارند که ممکن است تأثیر شدیدی بر مقیاس پذیری داشته باشند. از آنجاییکه بكمک این روش چندین کپی از یک منبع ایجاد می شود، اصلاح یک کپی باعث تفاوت آن با سایر کپی ها خواهد شد. در نتیجه، به کار گیری حافظه نهان و کپی برداری مشکلاتی را از نظر انسجام<sup>۱۰۰</sup> ایجاد می کنند.

میزان تحمل عدم انسجام<sup>۱۰۱</sup> تا حد زیادی بستگی به کاربری یک منبع دارد. بعنوان مثال، بسیاری از کاربران وب مشکلی ندارند که جستجوگر آنها سند موجود در حافظه نهان را بازگرداند که اعتبار<sup>۱۰۲</sup> آن در چند دقیقه اخیر بررسی نشده باشد. با این وجود، در بسیاری از موارد، از قبیل مزایده ها یا بورس اوراق بهادر الکترونیک، بایستی انسجام قوی تضمین شود. مشکل ناشی از انسجام قوی آن است که هرنوع بروزآوری بایستی فوراً به تمامی کپی های دیگر پخش شود. بعلاوه، چنانچه دو به روزآوری بصورت همزمان انجام شود، غالباً لازم است که هریک از کپی ها به یک ترتیب به روزآوری شوند. مواردی از این دست معمولاً مستلزم وجود نوعی مکانیزم همگام سازی جهانی است. متأسفانه، پیاده سازی این مکانیزم ها بصورتی مقیاس پذیر بسیار مشکل یا حتی غیرممکن است، چون فتونها و سیگنالهای الکتریکی بایستی از محدوده سرعت ۳۰۰ کیلومتر بر میلی ثانیه (یا همان سرعت نور) تبعیت کنند. در نتیجه، مقیاس پذیری از طریق کپی برداری ممکن است راه حل های ذاتاً غیرقابل مقیاس پذیری را معرفی کند. در فصل ۷ مجدداً به موضوع کپی برداری و انسجام بازخواهیم گشت.

با بررسی این تکنیک های مقیاس پذیری به راحتی می توان ادعا کرد که از نظر فنی، مقیاس پذیری اندازه<sup>۱۰۳</sup> کمترین مشکل را دارد. در بسیاری از موارد، صرفاً با افزایش ظرفیت ماشین می توان (حداقل بطور موقت و احتمالاً با صرف هزینه های زیاد) مشکل را حل کرد. در مقایسه، مقیاس پذیری جغرافیایی مشکل بزرگتری است چون می خواهیم مادر طبیعت را رام خود کنیم. در عمل ترکیب تکنیک های توزیع، کپی برداری و به کار گیری حافظه نهان با انواع مختلف انسجام، عملکرد مناسبی ارائه می دهد. نهایتاً اینکه مقیاس پذیری مدیریت اجرایی<sup>۱۰۴</sup>، تا حدی به دلیل نیاز ما به حل مشکلات غیرتکنیکی (از قبیل سیاستهای سازمانی و همکاری های انسانی)، مشکل ترین نوع مقیاس پذیری ها به نظر می رسد. با این وجود، صرفاً با نادیده گرفتن دامنه های مدیریت اجرایی، شاهد پیشرفت‌هایی در این زمینه بوده ایم. ایجاد و کاربرد گسترده فناوری نظیر به نظیر<sup>۱۰۵</sup> در دنیای امروز بیانگر چیزی است که با قرارگیری کنترل در اختیار کاربران نهایی قابل دستیابی می باشد (Aberer and Hauswirth،

<sup>100</sup> consistency

<sup>101</sup> inconsistency

<sup>102</sup> validity

<sup>103</sup> size scalability

<sup>104</sup> administrative scalability

<sup>105</sup> peer-to-peer technology

Lau, 2005 و گروه همکاران ۲۰۰۵ و Oram, ۲۰۰۱). با این وجود لازم به ذکر است که فناوری نظیر به نظری در بهترین حالت می تواند فقط یک راه حل نسبی برای حل مشکلات مقیاس پذیری در مدیریت اجرایی محسوب شود و به همین دلیل هنوز جای بحث و بررسی دارد.

## ۱-۲-۵- طرح اشکال

اکنون دیگر باید بدانید که توسعه سیستم های توزیعی می تواند واقعاً کار مشکلی باشد. همانطور که در بسیاری از موارد در سرتاسر این کتاب خواهیم دید، مسائل زیادی را باید به طور هم زمان در نظر گرفت و نتیجه آن افزایش پیچیدگی است. با این وجود، با تبعیت از چند اصل کلی در طراحی ها می توان سیستم های توزیعی را به گونه ای ایجاد کرد که همخوانی زیادی با اهداف ترسیم شده در این فصل داشته باشد. بسیاری از اصول از قوانین اساسی و صحیح مهندسی نرم افزار تبعیت کرده و به همین دلیل از تکرار آنها در اینجا خودداری می کنیم.

با این وجود، وجه تمایز سیستم های توزیعی با نرم افزار متداول، پراکندگی مؤلفه های آنها در سرتاسر شبکه است. لحاظ نکردن این پراکندگی در طراحی ها باعث پیچیدگی بیهوده بسیاری از سیستم ها و نهایتاً منجر به اشتباهاتی می شود که باید بعدها به دنبال راه چاره ای برای رفع آنها باشیم . پیتر دوج که زمانی در شرکت سان میکروسیستمز مشغول فعالیت بود، این اشتباهات را در قالب تصورات اشتباهی مدون و ارائه کرد که مبتدی ها را در اولین تلاش ها برای ایجاد برنامه های کاربردی توزیعی گرفتار خود می کند:

۱. شبکه قابل اطمینان<sup>۱۰۶</sup> است.
۲. شبکه ایمن<sup>۱۰۷</sup> است.
۳. شبکه همگن<sup>۱۰۸</sup> است.
۴. توپولوژی شبکه<sup>۱۰۹</sup> ثابت است.
۵. تأخیر<sup>۱۱۰</sup> صفر است.
۶. پهنای باند<sup>۱۱۱</sup> نامحدود است.
۷. هزینه انتقال<sup>۱۱۲</sup> صفر است.
۸. فقط یک مدیر اجرایی<sup>۱۱۳</sup> وجود دارد.

<sup>106</sup> reliable

<sup>107</sup> secure

<sup>108</sup> homogeneous

<sup>109</sup> network topology

<sup>110</sup> latency

<sup>111</sup> bandwidth

<sup>112</sup> transport cost

<sup>113</sup> administrator

به رابطه بین این فرضیات با ویژگی های مختص سیستم های توزیعی یعنی قابلیت وثوق، ایمنی، همگن بودن، توپولوژی شبکه، تأخیر، پهنانی باند، هزینه های انتقال و نهایتاً دامنه های مدیریتی توجه کنید. در هنگام ایجاد برنامه های کاربردی غیرتوزیعی بسیاری از این مشکلات اصلاً مطرح نمی شوند. اغلب اصول مورد بحث در این کتاب ارتباط نزدیکی با مفروضات بالا دارد. در تمام موارد، ما برای مشکلاتی راه حل ارائه می کنیم که در اثر نادرست بودن یک یا چند فرض بالا ایجاد شده اند. بعنوان مثال، چیزی به نام شبکه های قابل اطمینان به هیچ وجه وجود خارجی ندارد و بنابراین کسب شفافیت خرایی هم غیرممکن خواهد بود. یک فصل کامل کتاب را به این بحث اختصاص داده ایم که ارتباطات شبکه ای ذاتاً غیرایمن هستند. تاکنون استدلال آورده ایم که در سیستم های توزیعی مسئله ناهمگن بودن باید مدنظر قرار گیرد. در همین راستا، در مبحث مربوط به کپی برداری و حل مشکلات مقیاس پذیری، مفصلأً به مشکلات پهنانی باند و تأخیر اشاره خواهیم کرد. همچنین در موارد مختلف در ضمن بحث درباره مسائل مدیریتی، بطور ضمنی تصورات غلط در مورد صفر بودن هزینه انتقال و یکتا بودن دامنه های مدیریتی را مطرح می کنیم.

### ۱-۳- انواع سیستم های توزیعی

پیش از شروع بحث در مورد اصول سیستم های توزیعی، نگاه دقیتری به انواع مختلف سیستم های توزیعی خواهیم داشت. در ادامه، تفاوت بین سیستم های محاسبات توزیعی<sup>۱۱۴</sup>، سیستم های اطلاعات توزیعی<sup>۱۱۵</sup> و سیستم های نهفته توزیعی<sup>۱۱۶</sup> را تشریح خواهیم کرد.

#### ۱-۳-۱- سیستم های محاسبات توزیعی

یکی از مهم ترین گروه های سیستم های توزیعی در محاسبات بسیار کارآ استفاده دارد. در این رابطه دو زیرگروه می توان شناسایی کرد. در **محاسبه خوش**<sup>۱۱۷</sup>، سخت افزار زیربنایی متشکل از مجموعه ای از ایستگاههای کاری یا رایانه های شخصی (PCها) مشابه است که از طریق یک شبکه ناحیه محلی پرسرعت ارتباط نزدیکی با هم پیدا کرده اند. بعلاوه، تمامی گره ها سیستم عامل واحدی را اجرا می کند.

اما در مورد **محاسبه توری**<sup>۱۱۸</sup>، وضعیت تا حدودی متفاوت است. این زیرگروه متشکل از سیستم های توزیعی است که غالباً بشكل فدراسیونی از سیستم های کامپیوتری سازماندهی می شوند. هریک از

<sup>114</sup> distributed computing systems

<sup>115</sup> distributed information systems

<sup>116</sup> distributed embedded systems

<sup>117</sup> cluster computing

<sup>118</sup> grid computing

این سیستم ها ممکن است در دامنه مدیریت اجرایی متفاوتی قرار گرفته و حتی سخت افزار، نرم افزار و فن آوری شبکه بکاررفته در آنها بسیار با هم متفاوت باشد.

### سیستم های محاسبه خوشه ای

سیستم های محاسبه خوشه ای محصول بهبود نسبت قیمت به کارآیی کامپیوترهای شخصی و ایستگاههای کاری محسوب می شود. در مقطعی در گذشته، از هر دو بعد مالی و فنی امکان ساخت ابرکامپیوترها با استفاده از فن آوری در دسترس همگان<sup>119</sup> و صرفاً با قرار دادن مجموعه ای از کامپیوترهای نسبتاً ساده در یک شبکه پرسرعت فراهم شد. تقریباً در تمام موارد، از محاسبه خوشه ای برای برنامه نویسی های موازی<sup>120</sup> استفاده می شود که در آنها یک برنامه (با محاسبات فراوان) به صورت موازی روی چندین ماشین اجرا شود.

شکل ۱-۶- نمونه ای از سیستم های محاسبه خوشه ای(داخل شکل: گره اصلی<sup>121</sup>، برنامه کاربردی مدیریت، کتابخانه های موازی، سیستم عامل محلی، شبکه دسترسی از راه دور، گره محاسبه، مؤلفه برنامه کاربردی موازی، سیستم عامل محلی، گره محاسبه، مؤلفه برنامه کاربردی موازی، سیستم عامل محلی، شبکه استاندارد، شبکه پرسرعت).

عنوان یکی از نمونه های معروف کامپیوترهای خوشه ای می توان به خوشه های بیولف<sup>122</sup> براساس Linux اشاره کرد که پیکربندی کلی آن در شکل ۱-۶ ارائه شده است. هریک از خوشه ها متشکل از مجموعه ای از گره های محاسبه<sup>123</sup> است که بوسیله یک گره اصلی قابل کنترل و دسترسی هستند. این گره اصلی تخصیص گره ها به یک برنامه موازی خاص را هدایت کرده، دسته ای از کارهای محله در اختیار دارد و رابطی برای کاربران سیستم ایجاد می کند. به این ترتیب، گره اصلی واقعاً میان افزار لازم برای اجرای برنامه ها و مدیریت خوشه را اجرا می کند، در حالیکه گره های محاسبه غالباً فقط به یک سیستم عامل استاندارد نیاز دارند.

یکی از مهمترین بخش های این میان افزار از کتابخانه هایی برای اجرای برنامه های موازی تشکیل شده است. همانطور که در فصل ۴ هم خواهیم گفت، بسیاری از این کتابخانه ها عملأ فقط امکانات پیشرفتی ارتباطی براساس پیام ایجاد کرده ولی قادر به اداره فرآیند های معیوب، مسائل امنیتی و غیره نخواهند بود.

<sup>119</sup> Off-the-self

<sup>120</sup> parallel programming

<sup>121</sup> master node

<sup>122</sup> Beowulf

<sup>123</sup> compute nodes

بعنوان نمونه ای از این سازمان سلسله مراتبی می توان به روش متقارن<sup>۱۲۴</sup> مورد استفاده در سیستم های MOSIX (Amar, ۲۰۰۴) و گروه همکاران) اشاره کرد. MOSIX تلاش می کند تا شکل تک سیستمی<sup>۱۲۵</sup> از خوشه ارائه دهد؛ به این معنی که یک کامپیوتر خوشه ای با ظاهر به اینکه یک کامپیوتر منفرد است، شفافیت توزیعی بالایی را به فرآیند ارائه می دهد. همانطور که قبلاً هم ذکر کردیم، ایجاد چنین شکلی در همه شرایط غیرممکن است. در MOSIX، شفافیت توزیعی بالا با ایجاد امکان مهاجرت پویا<sup>۱۲۶</sup> و با قبضه<sup>۱۲۷</sup> فرآیندها در بین گره های تشکیل دهنده خوشه محقق می شود. مهاجرت فرآیند به کاربر امکان می دهد تا برنامه کاربردی را روی هریک از گره ها (که گره خانه<sup>۱۲۸</sup> نامیده می شود) آغاز کرده و پس از آن ، مثلاً جهت استفاده مفید و مؤثر از منابع، این گره را به صورتی شفاف به گره های دیگر انتقال دهد. در فصل ۳ مجدداً به موضوع مهاجرت فرآیند بازخواهیم گشت.

### سیستم های محاسبه توری

همگن بودن ، یکی از ویژگی های بارز محاسبه خوشه است. در اغلب موارد، کامپیوترهای داخل خوشه تاحد زیادی یکسان هستند، دارای سیستم عامل واحدی بوده و همگی از طریق یک شبکه واحد متصل می شوند. در حالیکه، سیستم های محاسبه توری تا حد زیادی ناهمگن هستند، یعنی هیچ فرضی در مورد سخت افزار، سیستم های عامل، شبکه ها، دامنه های مدیریت اجرایی، سیاستهای امنیتی و غیره وجود ندارد.

یکی از مهمترین مسائل در سیستم های محاسبه توری آن است که منابعی از سازمانهای مختلف در کنار هم قرار داده می شوند تا امکان تشریک مساعی گروه هایی از افراد یا مؤسسات فراهم شود. این نوع تشریک مساعی در قالب سازمان مجازی<sup>۱۲۹</sup> محقق می شود. افراد متعلق به یک سازمان مجازی حق دسترسی به منابعی تأمین شده برای همان سازمان را دارند. این منابع نوعاً شامل خدمتگزاران محاسبه ( از جمله ابرکامپیوترهایی که احتمالاً به صورت کامپیوترهای خوشه ای پیاده شده اند) ، امکانات ذخیره سازی و پایگاه های داده ای و در برخی موارد ، تعدادی از دستگاه های شبکه ای شده از قبیل تلسکوپ، حسگر و غیره می باشند .

به دلیل ماهیت این سیستم ها، قسمت اعظم نرم افزار مربوط به عملی کردن محاسبات توری، جهت ایجاد دسترسی به منابع مربوط به دامنه های مدیریت اجرایی مختلف و فقط برای کاربران و برنامه های کاربردی متعلق به همان سازمان مجازی تکامل یافته اند . به همین دلیل، عمدتاً روی مسائل معماری

<sup>124</sup> symmetric approach

<sup>125</sup> single-system image

<sup>126</sup> Dynamic

<sup>127</sup> Preemptive

<sup>128</sup> home node

<sup>129</sup> virtual organization

تمرکز می شود . نمونه ای از معماری های پیشنهادی (Foster و گروه همکاران) در شکل ۷-۱ ارائه شده است.

**شکل ۷-۱-۱- معماری لایه ای سیستم های محاسبه توری ( داخل شکل: برنامه های کاربردی، لایه جمعی، لایه اتصال، لایه منبع، لایه فابریک).**

این معماری از چهار لایه تشکیل شده است. پایین ترین لایه ، یعنی لایه فابریک<sup>۱۳۰</sup> ، رابط هایی را برای منابع محلی در محلی خاص بوجود می آورد. توجه داشته باشید که این رابط ها به گونه ای ساخته می شوند تا امکان اشتراک منابع در داخل یک سازمان مجازی فراهم شود . این رابط ها توابعی را برای پرس و جوی وضعیت و توانایی های منابع و همچنین توابعی برای مدیریت واقعی منابع (از قبیل قفل کردن منابع) ایجاد می کنند.

لایه اتصال<sup>۱۳۱</sup> متشکل از پروتکل های ارتباطی جهت پشتیبانی از تراکنش توری است که از چندین منبع استفاده می کنند . بعنوان مثال، برای انتقال داده ها در بین منابع یا صرفاً دسترسی به منابع از مکان دور نیازمند پروتکل هایی هستیم. بعلاوه، این لایه حاوی پروتکلهای امنیتی برای تأیید هویت کاربران و منابع است. توجه داشته باشید که در بسیاری از موارد هویت کاربران انسانی تأیید نمی شود؛ بلکه در عوض، برنامه هایی که به نمایندگی از جانب آنها عمل می کنند تأیید هویت می شوند . از این نقطه نظر، تفویض حقوق از کاربر به برنامه ها کارکرد مهمی است که باید در لایه اتصال پشتیبانی شود. در مبحث امنیت در شبکه های توزیعی مفصلًاً راجع به موضوع تفویض صحبت خواهیم کرد.

لایه منبع<sup>۱۳۲</sup> مسئول مدیریت در شبکه های منفرد است . این لایه با استفاده از توابع فراهم شده بوسیله لایه اتصال، رابط های فراهم شده بوسیله لایه فابریک را مستقیماً فراخوانی می کند. بعنوان مثال، این لایه کارکردهای مربوط به کسب اطلاعات پیکربندی روی یک منبع خاص، یا، در کل، برای انجام عملیاتهای خاصی از قبیل ایجاد فرآیند یا خوandن داده را ارائه می دهد. بنابراین لایه منبع مسئول کنترل دسترسی بوده و از اینرو، روی تأیید هویت صورت گرفته بعنوان بخشی از لایه اتصال تکیه می کند.

لایه بعد در سلسله مراتب بالا، لایه جمعی<sup>۱۳۳</sup> است. این لایه مسئول هدایت دسترسی به منابع متعدد بوده و نوعاً متشکل از سرویس هایی جهت کشف منبع، تخصیص و زمانبندی امور روی منابع متعدد، کپی سازی داده و غیره است. برخلاف لایه اتصال و منبع که از مجموعه استاندارد و نسبتاً کوچکی از پروتکلهای تشکیل می شود ، لایه اتصال ممکن است متشکل از پروتکلهای بسیار متفاوت برای اهداف بسیار متفاوت بوده و این خود بیانگر طیف وسیع سرویس هایی است که لایه مذکور می تواند به یک سازمان مجازی ارائه دهد.

<sup>130</sup> fabric layer

<sup>131</sup> connectivity layer

<sup>132</sup> resource layer

<sup>133</sup> collective layer

لایه برنامه کاربردی<sup>۱۳۴</sup> متشکل از برنامه های کاربردی است که در داخل یک سازمان مجازی عمل کرده و از محیط محاسبه توری<sup>۱۳۵</sup> استفاده می کنند.

لایه های جمعی، اتصال و منبع رویهم رفته هسته چیزی به نام لایه میان افزار توری<sup>۱۳۶</sup> را تشکیل می دهند. این لایه ها در مجموع امکان دسترسی و مدیریت منابعی را فراهم می کنند که بالقوه در سرتاسر سایتها می تواند پراکنده شده باشد. یکی از مهمترین مشاهدات از نقطه نظر میان افزاری این است که در محاسبه توری، همیشه ایده سایت (یا یک واحد مدیریت اجرایی) بچشم می خورد. این جهت گیری مشترک با تغییر موضع تدریجی به سمت **معماری سرویس محور**<sup>۱۳۷</sup> - که امکان دسترسی به لایه های مختلف را از طریق مجموعه ای از سرویس های وب فراهم می کند- تشدید شد (Joseph, ۲۰۰۴).

و گروه همکاران). تاکنون، این معماری منجر به تعریف معماری دیگری به نام **معماری سرویس های توری باز (OGSA)**<sup>۱۳۸</sup> شده است. در این شیوه معماری ، تنوع لایه ها و تعدد مؤلفه ها باعث ایجاد پیچیدگی می شود. به نظر می رسد که سرنوشت تمامی فرآیندهای استانداردسازی<sup>۱۳۹</sup> به پیچیدگی ختم می شود. جزئیات مربوط به OGSA را در کتاب (Foster, ۲۰۰۵) و گروه همکاران) مطالعه کنید.

## ۲-۳-۱- سیستم های اطلاعات توزیعی

یکی دیگر از گروههای مهم سیستم های توزیعی در سازمانهای مشاهده می شود که گرچه با مجموعه ای از برنامه های کاربردی شبکه ای شده سروکار دارند ، اما قابلیت کار با هم برای آنها تجربه ای بسیار سخت است. بسیاری از راه حلها میان افزاری موجود نتیجه کار با زیرساختی<sup>۱۴۰</sup> است که در آنها ادغام برنامه های کاربردی در یک سیستم اطلاعاتی شرکتی<sup>۱۴۱</sup> آسان تر بوده است.

ادغام<sup>۱۴۲</sup> در سطوح مختلفی قابل تشخیص است. در بسیاری از موارد، یک برنامه کاربردی شبکه ای شده شده فقط متشکل از خدمتگزار اجرا کننده آن برنامه کاربردی (در اغلب موارد مجهر به یک پایگاه داده ای) بود که در اختیار برنامه های از راه دوری بنام **مشتریان**<sup>۱۴۳</sup> قرار می گرفت. این مشتریان قادر به ارسال درخواست اجرای یک عملیات خاص برای خدمتگزار و متعاقب آن دریافت پاسخ بودند. ادغام در پایین ترین لایه به کاربران اجازه می داد تا تعدادی از درخواستها را ، که گاهی اوقات برای

<sup>134</sup> applications

<sup>135</sup> grid computing environment

<sup>136</sup> grid middleware layer

<sup>137</sup> service-oriented architecture

<sup>138</sup> open grid services architecture

<sup>139</sup> standardization process

<sup>140</sup> infrastructure

<sup>141</sup> enterprise-wide

<sup>142</sup> integration

<sup>143</sup> client

خدمتگزارهای مختلف بودند ، در قالب یک درخواست بزرگ بسته بندی کرده و تحت عنوان **تراکنش توزیعی**<sup>۱۴۴</sup> اجرا کند . ایده اصلی آن بود که یا تمام درخواستها اجرا شود و یا هیچیک از آنها . با پیشرفت تر شدن برنامه های کاربردی و تفکیک تدریجی آنها به مؤلفه های مستقل ( خصوصاً تمایز بین مؤلفه های پایگاه داده ای از مؤلفه های پردازشی ) ، مشخص شد که بهتر است ادغام از طریق ایجاد امکان ارتباط مستقیم برنامه های کاربردی با یکدیگر هم انجام شود. این ویژگی اکنون منجر به صنعت عظیمی به نام **ادغام برنامه کاربردی شرکتی (EAI)**<sup>۱۴۵</sup> شده است. در ادامه بیشتر راجع به این دو شکل از سیستم های توزیعی صحبت خواهیم کرد.

#### سیستم های پردازش تراکنش<sup>۱۴۶</sup>

برای روشن تر شدن بحث، روی برنامه های کاربردی پایگاه داده ای تمرکز می کنیم. در عمل، عملیات روی پایگاه داده ای معمولاً در قالب **تراکنش ها** انجام می شود. برنامه نویسی با استفاده از تراکنش ها مستلزم آشنایی با عمل های پایه ای<sup>۱۴۷</sup> خاصی است که یا باید بوسیله سیستم توزیعی زیربنایی<sup>۱۴۸</sup> تأمین شود و یا از طریق سیستم زمان اجرای زبان<sup>۱۴۹</sup>. برخی از نمونه های عمل های پایه ای تراکنش در جدول ۱-۸ ارائه شده است. لیست دقیق عمل های پایه ای بستگی به نوع اشیاء مورد استفاده در تراکنش دارد(Grey and Reuter, ۱۹۹۳). مثلاً در یک سیستم پستی، با عمل های پایه ای ارسال، دریافت و پیش بردن پست مواجه هستیم . اما در یک سیستم حسابداری، ممکن است عمل های پایه ای کاملاً متفاوتی به چشم بخورد . اما **READ** و **WRITE** از نمونه های معروف و متداول در اغلب سیستم ها هستند. جمله های عادی<sup>۱۵۰</sup>، فراخوانی های روال و امثال آن هم در تراکنش ها مجاز هستند. همانطور که در ادامه خواهیم گفت، فراخوانی های روال دور(RPCها) هم غالباً به شکل تراکنش بسته بندی شده و چیزی به نام **RPC تراکنشی**<sup>۱۵۱</sup> بوجود می آورد. در فصل ۴ مفصلأً راجع به RPCها صحبت خواهیم کرد.

عمل های پایه	شرح
BEGIN_TRANSACTION	آغاز تراکنش را علامتگذاری کن
END_TRANSACTION	تراکنش را خاتمه بده و تلاش کن متعهد شوی
ABORT_TRANSACTION	تراکنش را نابود کن و مقادیر قدیمی را بازگردان

<sup>۱۴۴</sup> distributed transaction

<sup>۱۴۵</sup> enterprise application integration

<sup>۱۴۶</sup> transaction processing systems

<sup>۱۴۷</sup> Primitives

<sup>۱۴۸</sup> underlying distributed system

<sup>۱۴۹</sup> language runtime system

<sup>۱۵۰</sup> ordinary statements

<sup>۱۵۱</sup> Transactional RPC

READ	داده ها را از یک فایل، جدول یا امثال آن بخوان
WRITE	داده ها را به یک فایل، جدول یا امثال آن بنویس

### شکل ۱-۸- نمونه ای از عمل های پایه در تراکنش ها

از TRANSACTION-BEGIN و TRANSACTION-END جهت مشخص کردن محدوده تراکنش ها استفاده شده و عملیات بین آنها بدنه تراکنش را تشکیل می دهد. ویژگی بارز تراکنش آن است که یا تمام این عملیاتها اجرا می شوند و یا هیچیک از آنها. عملیات ممکن است بسته به نوع پیاده سازی، فرآخوانی های سیستم، روالهای کتابخانه ای، یا جملات داخل پرانتز یک زبان باشند.

این ویژگی "یا همه چیز یا هیچ چیز" تراکنش ها یکی از چهار ویژگی عمدۀ تمام تراکنش ها است. به بیان دقیق‌تر، تراکنش ها دارای ویژگی های زیر هستند:

۱. اتمی (Atomic): از دید دنیای خارج، تراکنش بصورتی تفکیک ناپذیر انجام می شود.
۲. پایدار (Consistent): تراکنش ثابت های سیستم را مخدوش نمی کند.
۳. ایزوله (Isolated): تراکنش ها همزمان با یکدیگر تداخل ندارند.
۴. دائمی (Durable): تغییرات ناشی از تراکنش ها دائمی هستند.

گاهی اوقات این ویژگی ها با چسباندن حروف اول آنها ، در مجموع ACID (اسید) خوانده می شوند. اولین ویژگی مهم تمامی تراکنش ها اتمی بودن است. براساس این ویژگی، تمامی تراکنش ها یا بطور کامل ، و البته طی فقط یک اقدام فوری و تفکیک ناپذیر، انجام می شود و یا اصلاً رخ نمی دهد. در حین انجام فرآیندهای دیگر (چه خود آنها در تراکنش ها شرکت داشته باشند چه نه) قادر به مشاهده هیچیک از وضعیتهای میانی<sup>۱۵۲</sup> تراکنش نیستند.

دومین ویژگی پایداری تراکنش است. به این معنی که سیستم دارای ثابت هایی است که باید همواره برقرار باشند. بعنوان مثال، در سیستم های بانکداری، یکی از ثابت های مهم قانون حفاظت از پول است. پس از هر نقل و انتقال داخلی، مقدار وجه موجود در بانک باید معادل مقدار آن پیش از انتقال باشد. اما در صورت از بین رفتن ثبات در حین تراکنش ، حتی به صورت لحظه ای هم، این اشکال از دید دنیای بیرون کاملاً مخفی خواهد ماند.

ویژگی سوم به این معناست که تراکنش ها ایزوله یا قابلیت سری شدن<sup>۱۵۳</sup> دارند. به این معنا که چنانچه دو یا چند تراکنش به صورت همزمان در حال انجام باشند، نتیجه نهایی برای هریک از آنها و برای فرآیندهای دیگر، چنان است که گویی تمام تراکنش ها با ترتیب مشخص(وابسته به سیستم) انجام شده اند.

<sup>152</sup> intermediate states

<sup>153</sup> serialized

ویژگی چهارم دائمی بودن است. به این معنی است که تراکنش‌ها در صورت تعهد، بدون کوچکترین توجهی به رویدادها و اتفاقات جانبی، به پیش رفته و نتایج دائمی می‌شوند. پس از انجام تعهد، هیچ خرابی نمی‌تواند نتایج را ملغی کرده یا باعث مفقود شدن آنها شود. (در مورد ویژگی دائمی بودن در فصل ۸ صحبت خواهیم کرد).

تا به اینجا، تراکنش‌ها یک پایگاه داده‌ای منفرد تعریف شده است. مطابق شکل ۱-۹، تراکنش جاسازی شده<sup>۱۵۴</sup> از تعدادی زیرتراکنش تشکیل می‌شود. ممکن است در بالاترین سطح تراکنش با منشعب کردن فرزندانی<sup>۱۵۵</sup> که به صورت موازی باهم، اما روی ماشین‌های مختلف اجرا می‌شوند، باعث افزایش عملکرد یا تسهیل برنامه نویسی شود. هریک از این فرزندان هم ممکن است یک یا چند زیر تراکنش اجرا کرده و یا فرزندان خود را منشعب کنند.

**شکل ۱-۹ - نمایی از یک تراکنش جاسازی شده** (تراکنش جاسازی شده، زیرتراکنش، زیر تراکنش، پایگاه داده‌ای شرکت هواپیمایی، پایگاه داده‌ای هتل، دو پایگاه داده‌ای (مستقل) مختلف).

زیرتراکنش‌ها اشکالی کوچک اما مهم را مطرح می‌کند. فرض کنید که یک تراکنش چندین زیرتراکنش را به صورت موازی با هم راه اندازی و یکی از آنها را تعهد می‌کند و همین امر باعث قابل رؤیت شدن نتایج برای تراکنش والد<sup>۱۵۶</sup> می‌شود. با پیشروی محاسبات، پدر فرزند خود را قطع<sup>۱۵۷</sup> کرده و کل سیستم را به وضعیت پیش از آغاز تراکنش بالاترین لایه بازمی‌گرداند. بنابراین، نتایج زیرتراکنش تعهد شده باید خنثی<sup>۱۵۸</sup> شود. بنابراین مفهوم ماندگاری که در بالا گفته شد، فقط در مورد تراکنش‌ها بالاترین لایه<sup>۱۵۹</sup> صدق می‌کند.

از آنجاییکه تراکنش‌ها را می‌توان با عمق دلخواه جایگذاری کرد، برای انجام درست کارها نیازمند مدیریت اجرایی قابل ملاحظه‌ای هستیم. اما معانی مشخص هستند. با آغاز هر تراکنش‌ها یا زیرتراکنش، از نظر مفهومی کپی اختصاصی از تمامی داده‌های موجود در کل سیستم در اختیار آنها قرار داده می‌شود تا در صورت لزوم این داده‌ها را دستکاری کند. در صورت قطع شدن، دنیای خصوصی آن ناپدید می‌شود به طوری که گویا اصلاً وجود نداشته است. در صورت تعهد، این دنیای خصوصی جایگزین دنیای والد می‌شود. بنابراین، چنانچه یک زیر تراکنش تعهد شده و سپس زیر تراکنش جدیدی آغاز شود، زیرتراکنش<sup>۱۶۰</sup> اخیر نتایج تولیدی بوسیله زیرتراکنش اول را مشاهده خواهد

<sup>154</sup> nested transaction

<sup>155</sup> children

<sup>156</sup> parent transaction

<sup>157</sup> Abort

<sup>158</sup> undo

<sup>159</sup> top-level transactions

<sup>160</sup> subtransaction

کرد. به همین ترتیب، در صورت قطع شدن تراکنش (سطح بالاتر) در بر گیرنده سایر تراکنش‌ها تمامی زیرتراکنش‌ها آن هم قطع خواهند شد.

تراکنش‌ها جایگذاری شده، از آنجایی که روش طبیعی برای توزیع تراکنش در بین ماشین‌های مختلف بوجود می‌آورند، در سیستم‌های توزیعی دارای اهمیت هستند. این تراکنش‌ها از تقسیم‌بندی منطقی<sup>۱۶۱</sup> کار تراکنش اصلی تبعیت می‌کنند. مثلاً تراکنش مربوط به طرح ریزی سفری را که باید برای آن سه پرواز مختلف رزرو شوند، می‌توان منطقاً به سه زیرتراکنش تقسیم کرد. هریک از این زیرتراکنش‌ها بصورتی جداگانه و مستقل از دو تای دیگر قابل مدیریت است.

در اوائل ابداع سیستم‌های میان افزاری شرکتی<sup>۱۶۲</sup>، مؤلفه‌ای که کار تراکنش‌ها توزیعی (یا جایگذاری شده) را انجام می‌داد، در واقع هسته ادغام کننده برنامه‌های کاربردی در سطح خدمتگزار یا پایگاه داده ای را تشکیل می‌داد. این مؤلفه ناظر پردازش تراکنش<sup>۱۶۳</sup> یا بطور خلاصه ناظر TP خوانده می‌شد. وظیفه این مؤلفه، ایجاد امکان دسترسی برنامه‌های کاربردی به پایگاه‌های داده ای/ خدمتگزاران متعدد از طریق ارائه یک مدل برنامه نویسی تراکنشی به آن، مشابه شکل ۱-۱۰، بود.

شکل ۱-۱۰- نقش ناظر های TP در سیستم‌های توزیعی (داخل شکل: برنامه کاربردی مشتری، تراکنش، درخواست‌ها، پاسخ، ناظر TP، پاسخ، درخواست، پاسخ، خدمتگزار).

### ادغام برنامه کاربردی شرکتی

همانطور که قبل‌گفته شد، با افزایش تعداد برنامه‌های کاربردی جداسده از پایگاه‌های داده ای زیربنایی خود، لزوم وجود امکاناتی برای تلفیق برنامه‌های کاربردی بصورتی مستقل از پایگاه‌های داده ای آنها بیشتر آشکار شد. بخصوص اینکه، مؤلفه‌های برنامه کاربردی بایستی قادر به ارتباط مستقیم با یکدیگر باشند، و نه این که فقط از طریق رفتار درخواست/پاسخ که در سیستم‌های پردازش تراکنش مورد پشتیبانی است، با یکدیگر ارتباط داشته باشند.

همین نیاز به وجود ارتباط در میان برنامه‌های کاربردی باعث ایجاد مدل‌های ارتباطی مختلف شد که مفصل‌اً در همین کتاب مورد بحث قرار خواهند گرفت. مطابق شکل ۱-۱۱، ایده اصلی آن بود که برنامه‌های کاربردی موجود بتوانند مستقیماً با یکدیگر تبادل داده کنند.

شکل ۱-۱۱- میان افزار به صورت تسهیل کننده ارتباط<sup>۱۶۴</sup> در ادغام برنامه کاربردی شرکتی (داخل شکل: برنامه کاربردی مشتری، میان افزار ارتباطی، برنامه کاربردی طرف خدمتگزار).

<sup>161</sup> logical division

<sup>162</sup> enterprise middleware systems

<sup>163</sup> transaction processing monitor

<sup>164</sup> communication facilitator

انواع مختلفی از نرم افزارهای ارتباطی وجود دارند. در **فراخوانی های روال دور (RPC)**<sup>۱۶۵</sup>، مؤلفه های برنامه کاربردی می توانند با فراخوانی روال محلی، درخواستی را برای مؤلفه های دیگر برنامه کاربردی ارسال کند. در ادامه، درخواست به صورت یک پیام بسته بندی و برای فراخوان شده<sup>۱۶۶</sup> ارسال می شود. نتیجه هم بازگردانده شده و عنوان نتیجه فراخوانی روال به برنامه کاربردی عودت داده می شود.

همگام با افزایش مقبولیت فن آوری شیءگرا، تکنیکهایی هم برای فراخوانی های اشیاء از راه دور ایجاد شدند که امروزه تحت عنوان **فراخوانی های متدهای دور (RMI)**<sup>۱۶۷</sup> (RMI) شناخته می شود. RMI اساساً همان RPC، اما با این تفاوت جزئی است که بجای برنامه های کاربردی، روی اشیاء عمل می کند. نقطه ضعف RMI و RPC آن است که فراخواننده و فراخوان شده هردو باید در زمان ارتباط برقرار و در حال اجرا باشند. بعلاوه، باید دقیقاً از نحوه ارجاع به یکدیگر اطلاع داشته باشند. این وابستگی عمیق غالباً یک نقطه ضعف جدی تلقی شده و منجر به تولید چیزی به نام **میان افزار پیام گرا**<sup>۱۶۸</sup> یا بطور ساده **MOM** شده است. در این حالت، برنامه های کاربردی صرفاً پیامهایی را به نقاط تماس منطقی<sup>۱۶۹</sup> ارسال می کنند که غالباً بوسیله یک موضوع تعریف شده اند. به همین ترتیب، برنامه های کاربردی می توانند علاقه خود را به نوع خاصی از پیام اعلان کرده و پس از آن، میان افزار ارتباطی مراقب ارائه همان پیامها به آن برنامه های کاربردی خواهد شد. سیستم های موسوم به انتشار/اشترانک<sup>۱۷۰</sup> یکی از مهمترین و رویه گسترش ترین گروههای سیستم های توزیعی هستند. به همین دلیل بطور مفصل در فصل ۱۳ مورد بحث قرار گرفته اند.

### ۳-۳-۱- سیستم های فراگیر توزیعی

اغلب سیستم های توزیعی که تا به اینجا مورد بحث قرار گرفت، از ویژگی مشترکی بنام ثبات<sup>۱۷۱</sup> برخوردار بودند؛ به این معنا که گره ها ثابت بوده و اتصال کمایش باکیفیت و دائمی با شبکه برقرار می کنند. این ثبات تا حدودی از طریق روشهای مختلف مورد بحث در این کتاب -که هدف اغلب آنها کسب شفافیت توزیعی است- محقق می شود. عنوان مثال، با آگاهی از وجود انبوهی از تکنیکهای مربوط به مخفی سازی و بازسازی خرابی ها می توان گفت این تصور وجود دارد که خرابی ها اموری موردي هستند و به ندرت روی می دهند. به همین ترتیب، چون قادر به مخفی سازی مسائل مربوط به محل واقعی گره ها در شبکه بوده ایم، و به طور مؤثر این توهم را برای کاربران و برنامه های کاربردی ایجاد نماییم که گره ها در جای خود میخکوب هستند.

<sup>165</sup> remote method calls

<sup>166</sup> callee

<sup>167</sup> remote method invocations

<sup>168</sup> message-oriented middleware

<sup>169</sup> logical contact points

<sup>170</sup> publish/subscribe

<sup>171</sup> stability

با این وجود، با ابداع و عرضه دستگاه های محاسبه نهفته و متحرک امور تا حدی پیچیده تر شده است. اکنون دیگر با سیستم های توزیعی مواجه هستیم که بی ثباتی، رفتار پیش فرض<sup>۱۷۲</sup> آنها تلقی می شود. اغلب دستگاه های این سیستم های فراگیر توزیعی<sup>۱۷۳</sup> کوچک بوده، با باتری کار می کنند، متحرک بوده و فقط یک اتصال بی سیم دارند، اگرچه این ویژگی ها در همه دستگاه ها مشاهده نمی شوند. بعلاوه، این ویژگی ها را نباید لزوماً محدود کننده تلقی کرد، همچنان که در امکان پذیری تلفن های هوشمند مدرن دیده می شود (Roussos، ۲۰۰۵، و گروه همکاران).

سیستم های فراگیر توزیعی، همانطور که از نام آنها هم بر می آید، بخشی از محیط اطراف ما می باشد (و از همینرو، در اغلب موارد ذاتاً توزیعی هستند). یکی از مهمترین ویژگی این سیستم ها، فقدان عمومی کنترل مدیریت اجرایی انسانی در آنهاست. در بهترین حالت، این دستگاه ها بوسیله مالکانشان قابل پیکربندی هستند، ولی گذشته از این باید بصورتی اتوماتیک محیط خود را کشف کرده و به بهترین نحو ممکنه، "در آن جایگذاری شوند". این جایگذاری بوسیله Grimm و گروه همکاران (۲۰۰۴) با تعیین سه شرط زیر برای برنامه های کاربردی فراگیر فرموله شده است:

۱. شمول تغییرات بافتی<sup>۱۷۴</sup>.
۲. تشویق ترکیب موردنی<sup>۱۷۵</sup>.
۳. شناسایی اشتراک بعنوان پیش فرض.

شمول تغییرات بافتی به این معناست که یک دستگاه بایستی دائمًا از این حقیقت آگاه باشد که محیط آن هر لحظه دستخوش تغییر است. یکی از ساده ترین تغییرات آن است که مثلًا چون کاربر در بین ایستگاههای پایه در حال جابجایی است، شبکه دیگر در دسترسی نباشد. در این حالت، برنامه کاربردی بایستی از طریقی، مثلًا با ایجاد اتصال اتوماتیک با یک شبکه دیگر یا به هر صورت مناسب دیگر از خود واکنش نشان دهد.

ویژگی تشویق ترکیب موردنی اشاره به این واقعیت دارد که در سیستم های فراگیر، بسیاری از دستگاه ها به روش های بسیار متفاوت توسط کاربران متفاوت مورد استفاده قرار می گیرند. در نتیجه، پیکربندی مجموعه برنامه های کاربردی که روی یک دستگاه اجرا می شوند، چه بوسیله کاربر و چه از طریق موقعیت های میانی خود کار<sup>۱۷۶</sup> (اما تحت کنترل)، باید ساده باشد.

یکی از جنبه های بسیار مهم سیستم های فراگیر آن است که دستگاه ها غالباً با هدف دسترسی (احتمالاً ارائه) اطلاعات به سیستم ملحق می شوند. این امر لزوم ایجاد ابزاری جهت تسهیل خواندن،

<sup>172</sup> default behavior

<sup>173</sup> distributed pervasive systems

<sup>174</sup> contextual changes

<sup>175</sup> Ad hoc composition

<sup>176</sup> automated interpositions

ذخیره سازی، مدیریت و تشریک اطلاعات را مطرح می کند. به دلیل تناوب و تغییر اتصال دستگاه ها، فضایی که اطلاعات قابل دسترسی در آن وجود دارد، به احتمال قوی دائمًا تغییر خواهد کرد.

Mascolo و گروه همکاران (۲۰۰۴) و همچنین Latvakoski و Niemela (۲۰۰۴) به نتایج مشابهی دست یافتند: به دلیل شرایط متحرک بودن، دستگاه ها بایستی از انطباق آسان و وابسته به برنامه کاربردی به محیط محلی خود پشتیبانی کنند. آنها باید بصورتی مؤثر قادر به کشف سرویس ها و واکنش مناسب باشند. در مجموع، می توان نتیجه گرفت که شفافیت توزیعی در سیستم های فراگیر دیده نمی شود. در واقع، توزیع داده ها، فرآیندها و کنترل، ذاتی<sup>۱۷۷</sup> این سیستم ها بوده و به همین دلیل، شاید بهتر باشد که به جای تلاش برای مخفی سازی، این ویژگی را حتی در معرض دید قرار دهیم. حال بباید نگاهی به برخی از نمونه های واقعی سیستم های فراگیر داشته باشیم.

### سیستم های خانگی

بعنوان یکی از انواع سیستم های فراگیر که کاربرد روزافزونی هم پیدا کرده و در عین حال کمترین محدودیت را دارد، می توان به سیستم های مرتبط با شبکه های خانگی اشاره کرد. این سیستم ها گرچه معمولاً متشکل از یک یا چند کامپیوتر شخصی هستند، اما برخی لوازم الکترونیکی مصرفی از قبیل دستگاه های تلویزیون، تجهیزات صوتی و شکلی، دستگاه های بازی ، تلفن ها (هوشمند)، PDA ها و دیگر لوازم پوشیدنی شخصی را در یک سیستم واحد تلفیق می کنند. علاوه بر آن می توان انتظار داشت که در آینده انواع دستگاه ها از قبیل لوازم آشپزخانه، دوربین های نظارتی ، ساعت ها، کنترل کننده های روشنایی و غیره هم در یک سیستم توزیعی واحد ادغام شوند.

از دیدگاه سیستمی، پیش از محقق شدن سیستم های خانگی فراگیر چالش های زیادی باید در نظر گرفته شود. یکی از مهمترین مسائل این است که چنین سیستم هایی باید کاملاً خود-پیکربند<sup>۱۷۹</sup> و خود-مدیریت<sup>۱۸۰</sup> باشند. در صورتی که مؤلفه های این سیستم های خانگی توزیعی مستعد خطا باشند (که در بسیاری از دستگاه های جدید هم چنین است)، نمی توان انتظار داشت که کاربران نهایی علاقمند و قادر به نگهداری و استفاده از آنها باشند. استانداردهای **وصل کن و کار کن**<sup>۱۸۱</sup> جهانی (UPnP<sup>۱۸۲</sup>) تا حد زیادی این امر را محقق کرده اند؛ به طوری که دستگاه ها به صورت اتوماتیک آدرس IP بدهست می آورند، یکدیگر را کشف می کنند و بسیاری توانایی های دیگر (مجموع UPnP، ۲۰۰۳). اما هنوز هم مشکلاتی وجود دارد. بعنوان مثال، هنوز نمی دانیم چگونه می توان نرم افزار و سخت افزار این دستگاه ها را به راحتی و بدون دخالت انسانی به روزآوری کرد، یا این به روزآوری ها چه موقع انجام شوند تا سازگاری با دیگر دستگاه ها از بین نرود.

<sup>177</sup> inherent

<sup>178</sup> Personal Digital Assistant

<sup>179</sup> self-configuring

<sup>180</sup> self-managing

<sup>181</sup> Plug and Play

<sup>182</sup> Universal Plug and Play Standards

نکته مهم دیگر مدیریت چیزی به نام "فضای شخصی<sup>۱۸۳</sup>" است. با این فرض که سیستم های خانگی از تعداد زیادی دستگاه مشترک و همچنین شخصی تشکیل شده اند، و اینکه داده ها در سیستم های خانگی هم دچار محدودیت های اشتراک هستند، تلاش زیادی برای تحقیق این فضاهای شخصی صورت گرفته است. بعنوان مثال، بخشی از فضای شخصی آلیس شامل دستور کار<sup>۱۸۴</sup> او، عکس های خانوادگی، دفترچه خاطرات، موسیقی و تصاویر ویدئویی خریداری شده و غیره می باشد. دارایی های شخصی باید به صورتی ذخیره شوند که آلیس به محض نیاز به آنها دسترسی پیدا کند. بعلاوه، بخش هایی از این فضای شخصی (بطور موقع) باید بوسیله دیگران قابل دسترسی باشد، مثلًا برای گذاشتن قرارهای کاری. خوشبختانه مسائل ساده تر خواهند شد. مدت ها است که این تصور وجود دارد که فضاهای شخصی مربوط به سیستم های خانگی ذاتاً در سرتاسر دستگاه های مختلف توزیع شده اند. مشخص است که چنین پراکندگی می تواند منجر به مشکلات عدیده ای در روند همگام سازی شود. اما با افزایش سریع ظرفیت هارد دیسک ها در عین کاهش ابعاد آنها، می توان مشکلات را کاهش داد. پیکربندی یک واحد ذخیره سازی چند ترابایتی برای یک کامپیوتر شخصی مشکل واقعی نیست. همچنین در حال حاضر هارد دیسک های قابل جابجایی با ظرفیت صدها گیگابایت در داخل مدیاپلیر<sup>۱۸۵</sup> های قابل حمل نسبتاً کوچک جایگذاری می شوند. با افزایش مداوم ظرفیت ها، شاید در آینده شاهد استفاده از نوعی معماری در سیستم های فرآگیر خانگی باشیم که در آن یک ماشین منفرد (که در جایی در زیرزمین در نزدیکی سیستم گرمایش مرکزی مخفی شده) بعنوان رئیس عمل کرده و تمامی دستگاه های ثابت دیگر صرفاً نقش رابط دسترسی را برای انسانها بازی کنند. در اینصورت ممکن است دستگاه های شخصی برای اطلاعات لازم روزمره تحت فشار شدید قرار گیرند، ولی منبع ذخیره آنها هرگز به پایان نمی رسد.

با این وجود، در اختیار داشتن منبع ذخیره کافی حل مشکل مدیریت فضاهای شخصی محسوب نمی شود. بلکه توان ذخیره سازی مقادیر عظیم داده ها مسئله را به سمت ذخیره سازی داده های مربوط<sup>۱۸۶</sup> و پیدا کردن آنها برای استفاده های آتی سوق می دهد. هر روز بیش از پیش شاهد خواهیم بود که سیستم های فرآگیری از قبیل شبکه های خانگی، بكمک برنامه هایی به نام **توصیه کننده ها**<sup>۱۸۷</sup> (برنامه هایی که با برنامه های کاربران دیگر مشورت می کنند تا سلیقه های مشابه شناسایی شوند) راجع به محتوایی که باید در فضای شخصی افراد قرار گیرد، تصمیم گیری کنند. جالب است بدانید که مقدار اطلاعاتی که برنامه های توصیه کننده برای انجام وظیفه خود نیاز دارند، در اغلب موارد به اندازه کافی کوچک است به طوری که می توانند روی PDA ها نیز اجرا شوند (Miller, ۲۰۰۴، و گروه همکاران).

<sup>183</sup> personal space

<sup>184</sup> agenda

<sup>185</sup> Media player

<sup>186</sup> relevant data

<sup>187</sup> recommenders

## سیستم های الکترونیکی مراقبت از سلامت

یکی دیگر از مهمترین و رو به افزایش ترین انواع سیستم های فرآگیر ، سیستم های الکترونیکی مراقبت از سلامت (فردي) است. به دليل افزایش هزينه های درمانی، دستگاه های جدیدی برای نظارت بر سلامت افراد و برقراری تماس اتوماتیک با پزشکان در صورت نیاز ابداع شدند. يکی از مهمترین اهداف بسیاری از این سیستم ها جلوگیری از بستری شدن افراد در بیمارستان است.

سیستم های مراقبت از سلامت فردی غالباً مجهز به حسگرهای مختلفی هستند که به صورت یک شبکه (ترجیحاً بی سیم) سطح بدن<sup>۱۸۸</sup> (BAN) سازماندهی شده اند. مهم این است که این شبکه ها در بدترین حالت حداقل سختی را برای شخص داشته باشند. بنابراین شبکه باید بتواند علیرغم اتصال هیچ رشته (یعنی سیمی) به دستگاه های غیرمتحرک، در حین حرکت فرد باز هم عمل کند.

براساس شکل ۱-۱۲، در این رابطه می توان دو سازمان مشخص شناسایی کرد. در سازمان اول، یک هاب مرکزی<sup>۱۸۹</sup> بعنوان بخشی از BAN، داده های لازم را جمع آوری می کند. سپس این داده ها در فواصل زمانی به یک دستگاه ذخیره بزرگتر آفلود<sup>۱۹۰</sup> می شوند. مزیت طرح این است که هاب قادر به مدیریت BAN هم می باشد . در حالت دوم، BAN پیوسته و از طریق یک اتصال بی سیم، به یک شبکه خارجی مرتبط شده و داده های نظارتی را به آن ارسال می کند. برای مدیریت BAN، باید از تکنیک های مجازی استفاده شود. البته، ممکن است اتصالات دیگری هم با پزشک یا افراد دیگر ایجاد شود.

شکل ۱-۱۲- نظارت بر یک فرد در یک سیستم الکترونیکی فرآگیر مراقبت از سلامت با استفاده از (الف) هاب مرکزی و (ب) اتصال بی سیم پیوسته ( داخل شکل: حسگر یکبری شدن، حسگر ECG، حسگرهای حرکتی، شبکه سطح بدنی، ب) فرستنده، منبع ذخیره خارجی، GPRS/UMTS، شبکه سطح بدن).

از نقطه نظر سیستم های توزیعی، پرسشها یی از این قبیل مطرح می شود:

۱. داده های نظارت شده در کجا و چگونه باید ذخیره شوند؟
۲. چگونه می توان از مفقود شدن داده های حیاتی جلوگیری کرد؟
۳. برای تولید و پخش هشدارها نیازمند چه زیرساختی هستیم؟
۴. پزشکان چگونه می توانند بازخورد آن لاین<sup>۱۹۱</sup> ایجاد کنند؟
۵. چگونه می توان سیستم نظارتی بسیار مقاوم را محقق نمود؟
۶. مسائل امنیتی چه هستند و چگونه می توان سیاستهای مناسبی را به کار گرفت؟

<sup>188</sup> body-area networks

<sup>189</sup> central hub

<sup>190</sup> offload

<sup>191</sup> online feedback

برخلاف سیستم های خانگی، نمی توان انتظار داشت که معماری سیستم های فراغیر مراقبت از سلامت به سمت سیستم های تک خدمتگزاری تمایل پیدا کرده و در نتیجه دستگاه های ناظارتی با حداقل کار کرد عمل کنند. بلکه ، به دلایل بهره وری، دستگاه ها و شبکه های سطح بدن باید از پردازش داده های دورن شبکه ای<sup>۱۹۲</sup> پشتیبانی کنند؛ به این معنا که ، داده های ناظارتی بایستی، مثلاً ، پیش از ذخیره دائم یا ارسال به پزشک، متراکم شوند. برخلاف سیستم های اطلاعات توزیعی، هنوز هیچ پاسخ مشخصی برای این پرسشها ارائه نشده است.

### شبکه های حسگر

مثال آخر مربوط به سیستم های فراغیر، شبکه های حسگر است. در بسیاری از موارد، این شبکه ها بخشی از فن آوری توسعه فراغیری را تشکیل داده و ، همانطور که خواهیم دید، بسیاری از راه حلها مربوط به شبکه های حسگر به برنامه های کاربردی فراغیر باز می گردد. قابل توجه بودن شبکه های حسگر از نقطه نظر سیستم های توزیعی آن است که تقریباً در تمام موارد، جهت پردازش اطلاعات مورد استفاده قرار می گیرند . از این نظر، وظیفه آنها چیزی بیش از صرفاً تهیه سرویس های ارتباطی است که کار شبکه های کامپیوترا مرسوم است. در (Akyildiz ۲۰۰۲ و گروه همکاران) شرح کلی راجع به شبکه های حسگر از نقطه نظر شبکه ای و در (Zhao and Guibas ۲۰۰۴) از نقطه نظر سیستمی ارائه شده است . موضوع مرتبط دیگر ، شبکه های مش<sup>۱۹۳</sup> هستند که از مجموعه ای از گره های ثابت) تشکیل شده اند که از طریق پیوندهای بی سیم با هم ارتباط برقرار می کنند. این شبکه ها می توانند اساس بسیاری از سیستم های توزیعی مقیاس متوسط<sup>۱۹۴</sup> را تشکیل دهند که شرح کلی از آنها در (Akyildiz ۲۰۰۵ و گروه همکاران) آمده است .

شبکه های حسگر نوعاً از ده ها تا ده هزار گره نسبتاً کوچکی تشکیل شده اند که هریک مجهز به یک دستگاه حس کننده می باشد. اغلب شبکه های حسگر از ارتباط بی سیم<sup>۱۹۵</sup> استفاده کرده و انرژی گره ها غالباً از طریق باتری تأمین می شود. محدودیت منابع ، امکانات ارتباطی محدود و مصرف توان محدود باعث می شود تا در طراحی این شبکه ها موضوع بهره وری جایگاه ویژه ای داشته باشد.

در نظر گرفتن شبکه های حسگر به عنوان پایگاه های داده ای توزیعی، ارتباط آنها را با سیستم های توزیعی مشخص می کند. تصور این که بسیاری از شبکه های حسگر برای برنامه های کاربردی ناظارتی و اندازه گیری پیاده سازی شده اند (Bonnet و گروه همکاران)، سادگی و قابل فهم بودن این دیدگاه را نشان می دهد. در این موارد، اپراتور می تواند صرفاً با صدور پرس و جوهایی از قبیل: "بار ترافیکی به سمت شمال در بزرگراه ۱ چقدر است؟" اطلاعات را از (بخشی از) شبکه استخراج کند . این نوع پرس و جوها مشابه پرس و جوها در پایگاه های داده ای متعارف است. در این حالت، شاید لازم

<sup>192</sup> in-network data processing

<sup>193</sup> mesh networks

<sup>194</sup> medium-scale distributed systems

<sup>195</sup> wireless communication

باشد که پاسخ با تشریک مساعی تعداد زیادی از حسگرهای واقع در اطراف بزرگراه ۱ ارائه شده و بقیه حسگرهای به حال خود رها شوند.

مطابق شکل ۱-۱۳، برای سازماندهی شبکه های حسگر به صورت پایگاهی داده ای توزیعی دو راه وجود دارد. در روش اول، حسگرها تشریک مساعی نمی کنند، بلکه صرفاً داده های خود را به یک پایگاه داده ای متمرکز واقع در سایت اپراتور ارسال می کنند. در روش دوم، پرس و جوها به حسگرهای مربوطه ارسال شده و هر حسگر پاسخی را محاسبه می کند، البته اپراتور هم باید پاسخهای عودت داده شده را بصورتی منطقی متمرکز کند.

هیچیک از ایندو راه حل خیلی جالب نیستند. راه حل اول مستلزم آن است که حسگرها تمامی داده های اندازه گیری شده خود را از طریق شبکه ارسال کنند که البته اینکار باعث اتلاف انرژی و منابع شبکه خواهد شد. راه حل دوم هم می تواند باعث اتلاف شود، اگر توانایی متراکم سازی حسگرها و در نتیجه کاهش محسوس تعداد داده های بازگشتی به اپراتور ندیده گرفته شود. در واقع مشابه سیستم های فراغی مرآقبت از سلامت، در این شبکه ها هم نیازمند امکاناتی برای پردازش داده های داخل شبکه ای هستیم.

شکل ۱-۱۳- سازماندهی یک پایگاه داده ای شبکه حسگر در حین ذخیره سازی و پردازش داده (الف) فقط در سایت اپراتور یا (ب) فقط در حسگرها (داخل شکل: (الف) سایت اپراتور، داده های حسگر مستقیماً به اپراتور ارسال می شود، شبکه حسگر، (ب) سایت اپراتور، هریک از حسگرها قادر به پردازش و ذخیره داده ها هستند، شبکه حسگر، پرس و جو، حسگرها فقط پاسخهای ارسال می کنند).

پردازش داخل شبکه ای به اشکال مختلفی قابل انجام است. یک راه حل مشخص آن است که از طریق درختی که جامع تمام گره ها باشد، برای تمام گره های حسگرها یک پرس و جو ارسال کرده و سپس، نتایج را در حین پخش بازگشتی به ریشه -که محل آغاز آنها محسوب می شود- متراکم<sup>۱۹۶</sup> کنیم. تراکم در جایی رخ خواهد داد که دو یا چند شاخه از درخت با هم تلاقی می کنند. این طرح با وجود سادگی، سه پرسش عمده را مطرح می کند:

۱. چگونه می توان در یک شبکه حسگر (به صورتی پویا) یک درخت مفید و مؤثر ایجاد کرد؟
۲. متراکم سازی نتایج چگونه انجام می شود و آیا قابل کنترل است؟
۳. در صورت شکست پیوندهای شبکه چه اتفاقی می افتد؟

این پرسشها تا حدودی در TinyDB – که رابطی توصیفی<sup>۱۹۷</sup> (پایگاه داده ای) را برای شبکه های حسگر بی سیم پیاده می کند- پاسخ داده شده اند. بطور خلاصه، TinyDB می تواند از هر نوع

<sup>196</sup> aggregate

<sup>197</sup> declarative

الگوریتم مسیردهی براساس درخت استفاده کند. یکی از گره های میانی نتایج را از بچه های خود جمع آوری کرده و به همراه یافته های خود متراکم کرده و به سمت ریشه ارسال می کند . برای افزایش اثربخشی ، پرس وجوها با تعریف یک دوره زمانی ، امکان زمانبندی دقیق عملیات و در نتیجه، استفاده بهینه از منابع و انرژی شبکه را فراهم می کنند . جزئیات بیشتر را در (Madden، ۲۰۰۵) و گروه همکاران) مطالعه کنید.

اما اگر بتوان پرس و جوها را از نقاط مختلف در شبکه آغاز کرد، استفاده از درختهای تک ریشه ای از قبیل TinyDB شاید چندان هم مؤثر نباشد. در روش دیگر، شبکه های حسگر به گره های ویژه ای مجهز می شوند که نتایج و همچنین پرس وجوهای مربوط به آن نتایج به آنها ارسال می شوند . عنوان یک نمونه ساده، پرس وجوها و نتایج مربوط به خواندن درجه حرارت در نقطه ای متفاوت نسبت به نقطه اندازه گیری رطوبت جمع آوری می شوند. این روش مستقیماً مشابه ایده سیستم های انتشار / اشتراک است که در فصل ۱۳ مورد بحث قرار خواهد گرفت.

#### ۱-۴ - خلاصه

سیستم های توزیعی از کامپیوترهای خودمختاری تشکیل شده اند که ضمن همکاری با هم، نمایی از یک سیستم منسجم و منفرد، ارائه می دهند. یکی از مهم ترین مزایای این دست سیستم ها آن است که تلفیق برنامه های کاربردی مختلف را ، که روی کامپیوترهای مختلف هم در حال اجرا هستند، در یک سیستم واحد تسهیل می کنند. مزیت دیگر اینکه سیستم های توزیعی در صورت طراحی مناسب، به خوبی با ابعاد شبکه زیربنایی مقیاس برداری می شوند . اما هزینه ای که باید در قبال این مزایا بپردازیم ، افزایش پیچیدگی نرم افزار، افت کارآیی و کاهش سطح امنیتی است. با وجود تمام این اشکالات، هنوز هم علاقه زیادی به ساخت و نصب سیستم های توزیعی در سرتاسر جهان وجود دارد.

هدف غالب سیستم های توزیعی، مخفی سازی بسیاری از پیچیدگی های مربوط به توزیع فرآیندها، داده و کنترل است. اما کسب این شفافیت توزیعی نه تنها باعث افت عملکرد می شود، در موقعیتهاي عملی هم هرگز به طور کامل محقق نمی شود . باید در طراحی سیستم های توزیعی، مسئله ایجاد توازن در کسب اشکال مختلف شفافیت توزیعی لاحظ شده و همین امر درک آنها را پیچیده می کند.

پیچیدگی بیشتر ناشی از این واقعیت است که بسیاری از سازندگان در ابتدای کار فرضیات اساساً نادرستی راجع به شبکه زیربنایی در نظر دارند . بعد ها که این فرضیات با شکست مواجه می شود، ممکن سرپوش گذاشتن بر رفتار ناخواسته ناشی از آنها مشکل ساز شود. عنوان مثال، این فرض که تأخیرهای شبکه ناچیز هستند را در نظر بگیرید. بعداً، حین انتقال سیستم موجود به یک شبکه حوزه گسترده، مخفی سازی تأخیرها ممکن است تأثیر شدیدی بر طرح اولیه سیستم داشته باشد. از نمونه فرض های نابجای دیگر می توان به فرض قابل اطمینان بودن ، ثبات ، ایمنی و همگن بودن شبکه اشاره کرد.

انواع مختلف سیستم های توزیعی را می توان در سه گروه سیستم های پشتیبان محاسبات، پردازش اطلاعات و فراغیری دسته بندی کرد. سیستم های محاسبه توزیعی نوعاً برای برنامه های کاربردی با کارآیی بالا ایجاد شده اند که از حوزه محاسبه موازی سرچشم می گیرند. گروه دیگری از سیستم های توزیعی را می توان در محیط های دفاتر کار سنتی مشاهده کرد که پایگاههای داده ای در آنها نقش مهمی را ایفا می کنند. معمولاً در این محیط ها از سیستم های پردازش تراکنش<sup>۱۹۸</sup> استفاده می شود. در آخرین گروه سیستم های توزیعی نوظهور، مؤلفه ها کوچک بوده و سیستم به صورت موردي ساخته شده است، اما مدیریت آنها دیگر بر عهده مدیر اجرایی سیستم نمی باشد. نمونه بارز این گروه، محیط های محاسبه همه جا حاضر<sup>۱۹۹</sup> شناخته می شود.

## مسائل فصل

- ۱- یکی دیگر از تعاریفی که برای سیستم های توزیعی ارائه می شود به این صورت است: "مجموعه ای از کامپیوترهای مستقل که مانند یک سیستم واحد بنظر می رسد. یعنی حتی این که چندین کامپیوتر هستند از دید کاربران مخفی است". نمونه ای از مواردی را ذکر کنید که این دیدگاه برای آن بسیار کاربردی است.
- ۲- راجع به نقش میان افزارها در سیستم های توزیعی توضیح دهید.
- ۳- بسیاری از سیستم های شبکه ای شده به صورت یک دفترکار عقبی<sup>۲۰۰</sup> و یک دفتر کار جلویی<sup>۲۰۱</sup> سازمان دهی شده اند. سازمانها چگونه با نمای منسجم سیستم های توزیعی هماهنگ می شوند؟
- ۴- راجع به معنی شفافیت (توزیعی) توضیح داده و نمونه هایی از انواع مختلف شفافیت ذکر کنید.
- ۵- توضیح دهید که چرا در برخی موارد مخفی سازی وقوع و ترمیم خرابی در سیستم های توزیعی بسیار سخت است؟
- ۶- چرا کسب حداکثر درجه شفافیت ممکنه همواره ایده خوبی نیست؟
- ۷- سیستم توزیعی باز را تشریح کرده و مزایای باز بودن را توضیح دهید؟
- ۸- دقیقاً معنای سیستم های مقیاس پذیر توضیح دهید.
- ۹- مقیاس پذیری به روشهای مختلف قابل دستیابی است. این روش ها چیستند؟
- ۱۰- سازمان های مجازی را توضیح داده و اشاره ای به نحوه پیاده سازی این دست سازمانها داشته باشید.

<sup>198</sup> transaction processing

<sup>199</sup> ubiquitous

<sup>200</sup> Back office

<sup>201</sup> Front office

- ۱۱- گفتیم که در صورت سقط شدن تراکنش ، دنیا طوری به وضعیت سابق خود باز می گردد که گویا اصلاً تراکنشی انجام نشده بود . اما دروغ گفتیم. نمونه ای از مواردی ذکر کنید که بازگرداندن دنیا به صورت قبل غیر ممکن باشد.
- ۱۲- اجرای تراکنشهای جایگذاری شده مستلزم وجود شکلی از هماهنگی است. راجع به وظیفه واقعی یک هماهنگ کننده<sup>۲۰۲</sup> توضیح دهید.
- ۱۳- گفتیم که شفافیت توزیعی ممکن است در سیستم های فرآگیر موجود نباشد. اما این گفته در مورد همه انواع شفافیتها صدق نمی کند . یک مثال ذکر کنید.
- ۱۴- در متن درس نمونه هایی از سیستم های توزیعی فرآگیر از قبیل سیستم های خانگی، سیستم های الکترونیک مراقبت از سلامت و شبکه های حسگر ذکر کردیم . نمونه های دیگری را از این مجموعه نام ببرید.
- ۱۵- (تکلیف آزمایشگاهی) طرحی برای یک سیستم خانگی متشکل از یک خدمتگزار رسانه مجرزا با امکان اتصال یک مشتری بی سیم به آن ترسیم کنید . این مشتری به تجهیزات صوتی/ تصویری (آنالوگ) متصل بوده و جریانات رسانه ای دیجیتال را به خروجی آنالوگ تبدیل می کند. خدمتگزار روی ماشین جداگانه ای اجرا می شود که احتمالاً به اینترنت متصل است ، اما هیچ صفحه کلید و/ یا صفحه نمایشی به آن متصل نیست.

---

<sup>202</sup> coordinator

## معماری

سیستم های توزیعی غالباً قطعات پیچیده نرم افزاری هستند که براساس تعریف مؤلفه<sup>۲۰۳</sup> های آنها در سرتاسر ماشینهای مختلف پراکنده شده اند. جهت فائق آمدن بر پیچیدگی این سیستم ها، باید آنها را به نحو مناسبی سازماندهی نمود. روشهای متعدد و مختلفی برای بررسی نحوه سازماندهی سیستم های توزیعی وجود دارد. اما یک روش مشخص، تمایز قائل شدن میان سازمان منطقی مجموعه مؤلفه های نرم افزاری و از طرف دیگر، درک فیزیکی واقعی است.

چگونگی سازمان سیستم های توزیعی تا حد زیادی در رابطه با مؤلفه های نرم افزاری تشکیل دهنده سیستم است. این **معماری های نرم افزار** بیانگر نحوه سازماندهی مؤلفه های مختلف نرم افزاری و تقابل آنهاست. در این فصل، ابتدا اشاره ای خواهیم داشت به برخی از پرکاربردترین روش هایی که غالباً جهت سازماندهی سیستم های کامپیوترا (توزیعی) مورد استفاده قرار می گیرد.

تحقیق یک سیستم توزیعی مستلزم آن است که مؤلفه های نرم افزار ایجاد و روی ماشین های واقعی قرار داده شوند. برای انجام اینکار روشهای مختلفی وجود دارد. نحوه استقرار نهايی **معماری نرم افزار** غالباً **معماری سیستم** هم خوانده می شود. در این فصل ابتدا نگاهی خواهیم داشت به معماری های متمرکز مرسوم که در آنها، پیاده سازی اغلب مؤلفه های ( و بنابراین کارکردهای) نرم افزاری بر روی یک سرور واحد قرار دارد و مشتری های دور قادر هستند تا با روشهای ساده ارتباطی به آن سرور دسترسی پیدا کنند. سپس، معماری های غیرمتمرکز را بررسی خواهیم کرد که در آنها، چندین ماشین نقشهای کمابیش یکسانی ایفا کرده و همچنین نگاهی خواهیم داشت به سازمانهای تلفیقی.

همانطور که در فصل ۱ توضیح داده شد، یکی از اهداف مهم سیستم های توزیعی ، تفکیک برنامه های کاربردی از بسترها زیربنایی از طریق ایجاد یک لایه میان افزاری است. استفاده از چنین لایه ای یکی از مهمترین تصمیمات معماری محسوب شده و هدف اصلی آن، ایجاد شفافیت توزیعی است. بالین وجود، دستیابی به این شفافیت مستلزم ایجاد برخی موازن<sup>۲۰۴</sup> ها است. همین امر منجر به تنوع روشهای تطبیق پذیر<sup>۲۰۵</sup> در میان افزارها شده است. از آنجاییکه این روشهای بر سازمان میان افزار تأثیر می گذارند، راجع به پرکاربردترین آنها در همین فصل صحبت خواهیم کرد.

روش دیگر برای ایجاد انطباق پذیری در سیستم های توزیعی ، ایجاد امکان نظارت بر رفتار برای خود سیستم و اتخاذ اقدامات مناسب در صورت لزوم است. این دیدگاه موجب تولید انواعی از سیستمهای نام **سیستم های خودکار**<sup>۲۰۶</sup> شده است. این دست سیستم های توزیعی معمولاً به شکل حلقه کنترل

<sup>203</sup> Component

<sup>204</sup> Trade-off

<sup>205</sup> Adaptive

<sup>206</sup> Automatic

بازخوردی سازماندهی شده و یکی از عوامل مهم معماری در زمان طراحی سیستم می باشند. در این فصل، بخش مستقلی را به سیستم های توزیعی خودکار اختصاص داده ایم.

## ۱-۲- شیوه های معماری<sup>۲۰۷</sup>

بحث خود در مرور معماري ها را ابتدا با بررسی سازمان منطقی سیستم های توزیعی به مؤلفه های نرم افزاری یا همان معماري نرم افزار آغاز می کنیم (۲۰۰۳، Bass و گروه همکاران). تحقیق در مورد معماري های نرم افزارها به میزان قابل ملاحظه ای پیشرفته ای پیشگفت کرده و امروزه دیگر اغلب باور دارند که طراحی یا استفاده از نوع خاصی از معماري در توسعه موفقیت آمیز سیستم های بزرگ اهمیت کلیدی دارد.

در بحث حاضر مفهوم شیوه معماري اهمیت به سزا دارد. چنین شیوه ای شامل مؤلفه ها، نحوه اتصال مؤلفه ها به یکدیگر، تبادل داده ها در بین مؤلفه ها و در نهايیت، نحوه پیکربندی اين عوامل برای تشکيل يك سیستم واحد است. منظور از **مؤلفه واحد** پیمانه ای<sup>۲۰۸</sup> با واسطه<sup>۲۰۹</sup> های کاملاً تعریف شده لازم و موجود است که در داخل محیط خود قابل تعویض هستند (OMG, ۲۰۰۴b). همانطور که ذیلاً هم خواهیم گفت، نکته مهم در مورد يك مؤلفه برای سیستم های توزیعی آن است که مشروط به لحاظ کردن واسطه آن، مؤلفه قابل تعویض است. يکی از مفاهیمی که درک آن تاحدودی مشکل است، اتصال گر<sup>۲۱۰</sup> ها هستند که غالباً به صورت مکانیزمی برای میانجی گری بین ارتباطات، هماهنگ سازی، یا همکاری در بین مؤلفه ها تعريف می شوند (Shaw and Mehta, ۱۹۹۷؛ Clements, ۲۰۰۰). به عنوان مثال، می توان با استفاده از امکانات مربوط به فراخوانی روال ها (از راه دور)، تبادل پیام، یا جریان های داده ای یک اتصال گر ساخت.

بالاستفاده از مؤلفه ها و اتصال گرها ها می توان پیکربندی های مختلفی ایجاد نمود که نهایتاً در قالب شیوه های معماري دسته بندی می شوند. شیوه های مختلفی شناسایی شده اند که از مهمترین آنها در سیستم های توزیعی می توان به موارد زیر اشاره کرد:

۱. معماري های لایه ای<sup>۲۱۱</sup>
۲. معماري های براساس شی<sup>۲۱۲</sup>
۳. معماري های داده محور<sup>۲۱۳</sup>
۴. معماري های بر اساس رویداد<sup>۲۱۴</sup>

<sup>207</sup> Architectural Styles

<sup>208</sup> Modular

<sup>209</sup> Interface

<sup>210</sup> Connector

<sup>211</sup> Layered

<sup>212</sup> Object-based

<sup>213</sup> Data-centered

<sup>214</sup> Event-based

ایده اساسی در شیوه لایه ای بسیار ساده است: مؤلفه ها به صورت لایه ای سازماندهی می شوند . همانطور که در شکل (الف) ۲-۱ مشاهده می کنید، مؤلفه واقع در لایه  $L_1$  می تواند مؤلفه های واقع در لایه زیرین آن  $L_1$  (و نه برعکس) را فراخوانی نماید. این مدل بطور گسترده در جوامع شبکه ای به کار برده شده و در فصل ۴ شرح مختصری از آنرا ارائه خواهیم کرد. یکی از مشاهدات کلیدی این است که کنترل غالباً از یک لایه به لایه دیگر جریان می یابد. به بیان دیگر، درخواستها بصورت سلسله مراتبی از بالا به پایین و نتایج از پایین به بالا جریان پیدا می کنند.

در حالیکه در معماری های براساس شیء از ساختار بسیار آزادانه تر شکل (ب) ۲-۱ تبعیت می شود. بطور خلاصه، هر شیء متناظر با چیزی است که ما آنرا مؤلفه می نامیم و این مؤلفه ها از طریق یک مکانیزم فراخوانی روال (از راه دور) به هم متصل می شوند. تعجبی ندارد اگر بگوییم که این معماری نرم افزاری با معماری سیستم مشتری-سرور که قبلاً گفته شد، هماهنگی دارد. معماری های لایه ای و براساس شیء هنوز هم مهمترین شیوه های سیستم های بزرگ نرم افزاری محسوب می شوند (۲۰۰۳، Bass و گروه همکاران).

**شکل ۲-۱- شیوه معماری (الف) لایه ای و (ب) براساس شیء (داخل شکل: لایه N، لایه ۱، لایه ۲، لایه ۱، لایه ۲، لایه ۱، شیء، فراخوانی روش).**

معماری های داده محور<sup>۲۱۵</sup> براساس ایده برقراری تماس بین فرآیندها از طریق یک منبع مشترک (فعال یا غیرفعال) شکل گرفته اند . می توان استدلال کرد که اهمیت این معماری ها در سیستم های توزیعی به اندازه معماری های لایه ای و شیء محور است. بعنوان مثال، مجموعه بزرگی از برنامه های کاربردی تحت شبکه براساس سیستم فایل توزیعی اشتراکی بوجود آمده اند که در آنها ، تقریباً تمام ارتباطات از طریق فایلها انجام می شود. به همین ترتیب، سیستم های توزیعی براساس وب، که به تفصیل در فصل ۱۲ مورد بحث قرار خواهد گرفت، تا حد زیادی داده محور هستند. یعنی در آنها فرآیندها با استفاده از سرویسهای داده ای مشترک براساس وب ارتباط برقرار می کنند.

در حالت معماری براساس رویداد<sup>۲۱۶</sup>، فرآیندها اساساً از طریق انتشار رویدادها ارتباط برقرار کرده و مطابق شکل (الف) ۲-۲، بصورت اختیاری داده هم حمل می کنند. در سیستم های توزیعی، پخش رویداد غالباً با آنچه به نام سیستم های انتشار / اشتراک<sup>۲۱۷</sup> خوانده می شود (Eugster، ۲۰۰۳، و گروه همکاران)، مرتبط است. ایده اساسی این است که فرآیندها، در صورتی اقدام به پخش رویدادها خواهند کرد که میان افزار اطمینان دهد که فقط فرآیندهای قبلاً مشترک شده در آن رویداد آنرا دریافت می کنند. مزیت عمدی سیستم های براساس رویداد آن است که فرآیندها دارای اتصال سست<sup>۲۱۸</sup>

<sup>215</sup> Data-Centric

<sup>216</sup> Event-based

<sup>217</sup> Publish/Subscribe

<sup>218</sup> Loosely coupled

هستند و در مجموع، لزومی ندارد که آشکارا به یکدیگر ارجاع دهند. این حالت به نام بازشدن اتصال در فضا یا به بیان دیگر ، بازشدن اتصال مرجعی<sup>۲۱۹</sup> هم خوانده می شود.

شکل ۲-۲- شیوه معماری (الف) براساس رویداد و (ب) فضای داده ای اشتراکی (داخل شکل: مؤلفه، تحويل رویداد، گذرگاه رویداد، انتشار، مؤلفه، ب) مؤلفه، تحويل داده، انتشار، فضای داده ای ( دائم ) اشتراکی).

معماری های براساس رویداد را می توان با معماری های داده محور تلفیق کرد و چیزی به نام فضاهای داده ای اشتراکی<sup>۲۲۰</sup> ایجاد نمود . بطور خلاصه ، معنای فضاهای داده ای اشتراکی آن است که فرآیندها اکنون از نظر زمانی هم به یکدیگر اتصال ندارند و دیگر لازم نیست که هردو در حین انجام ارتباط فعال باشند. بعلاوه، بسیاری از فضاهای داده ای اشتراکی از یک رابط شبکه SQL برای منبع اشتراکی استفاده می شود تا بتوان - مشابه حالت فایلها- با استفاده از توصیف ونه یک مرجع آشکار، به داده ها دسترسی پیدا کرد. به دلیل اهمیت موضوع کل فصل ۱۳ را به این شیوه ویژه معماری اختصاص داده ایم.

وجه اهمیت این معماری های نرم افزاری در سیستم های توزیعی آن است که هدف همه آنها دستیابی به شفافیت توزیعی (در حدی معقول و منطقی) است. با این وجود، همانطور که قبلًا هم بحث کردیم، شفافیت توزیعی مستلزم ایجاد موازن<sup>۲۲۱</sup> هایی میان کارآیی، تحمل خرابی، سهولت برنامه نویسی و غیره است. از آنجایی که هیچ راه حل منحصر به فردی وجود ندارد که در شرایط تمامی برنامه های کاربردی توزیعی ممکن صدق کند، محققان دیگر به دنبال ایجاد سیستم توزیعی واحدی نیستند که در ۹۰٪ تمامی حالات ممکن قابل کاربرد باشد .

## ۲-۲- معماری های سیستم

حال که تاحدوی با برخی از شیوه های متعارف معماری آشنا شدیم ، اجازه دهید نگاهی داشته باشیم به آمار واقعی سیستم های توزیعی که براساس محل قرارگیری مؤلفه های نرم افزاری سازمان داده شده اند. تصمیم گیری درمورد مؤلفه های نرم افزاری ، تعامل آنها و محل قرارگیری آنها منجر به حالتی از معماری نرم افزاری می شود که به نام **معماری سیستم هم** معروف است ( Bass، ۲۰۰۳ و گروه همکاران). در ادامه راجع به سازمانهای متمنکز و غیرمتمنکز و انواع مختلف ترکیب آنها صحبت خواهیم کرد.

### ۲-۲-۱- معماری های متمنکز

علیرغم نبود اتفاق نظر در مورد بسیاری از مسائل سیستم های توزیعی، بسیاری از محققان و صاحبنظران فن برسر این مسئله توافق کرده اند که این دیدگاه که مشتریان از سرورها تقاضای سرویس

<sup>21۹</sup> Referentially decoupled

<sup>22۰</sup> Shared data spaces

<sup>22۱</sup> Trade-off

می کنند، به ما در درک و اداره پیچیدگی سیستم های توزیعی کمک کرده و دیدگاه مفیدی محسوب می شود.

در مدل اصلی مشتری- سرور، فرآیندها در سیستم توزیعی به دو گروه (احتمالاً دارای هم پوشانی) تقسیم می شوند: سرور، فرآیند اجراکننده یک سرویس خاص از قبیل سرویس سیستم فایل یا سرویس پایگاه داده ای است. در حالیکه مشتری، فرآیند درخواست کننده یک سرویس خاص از سرور از طریق ارسال درخواست برای سرور و سپس، انتظار برای دریافت پاسخ از جانب آن است. تقابل مشتری- سرور که رفتار درخواست- پاسخ هم نامیده می شود، در شکل ۲-۳ نمایش داده شده است.

**شکل ۲-۳ - تعامل عمومی بین مشتری و سرور( داخل شکل: مشتری، انتظار برای نتیجه، درخواست، پاسخ، سرور، ارائه سرویس، زمان).**

ارتباط بین مشتری و سرور را می توان در صورت مطمئن بودن شبکه زیربنایی- که درمورد بسیاری از شبکه های محلی صدق می کند- با استفاده از یک پروتکل بدون اتصال ساده پیاده نمود. در این موارد، هنگامیکه مشتری سرویس خاصی را تقاضا می کند، پیامی را برای سرور بسته بندی می کند. داخل پیام سرویس مورد نظر مشخص شده و داده های ورودی نیز قرار داده شده اند. سپس پیام مذکور برای سرور ارسال می شود. سرور که به نوبه خود همواره درحال انتظار برای درخواست جدید قرار دارد، پیام را پردازش ، نتایج را به شکل پیام پاسخ بسته بندی و در نهایت برای مشتری ارسال می نماید. مزیت آشکار استفاده از پروتکل بدون اتصال ،کارآمدی است. مادامی که پیام ها مفقود یا معیوب نشوند، پروتکل درخواست/ پاسخ فوق به خوبی عمل خواهد کرد. متأسفانه، مقاوم سازی پروتکلها در برابر خرابی های موردنی سیستم ارسال چندان هم کار ساده ای نیست. تنها راه حل ممکنه شاید این باشد که به مشتری اجازه دهیم تا در صورت عدم دریافت پیام پاسخ، درخواست خود را مجدد ارسال نماید. اما مسئله این است که مشتری نمی تواند تشخیص دهد که آیا مشکل در اثر مفقود شدن پیام درخواست اصلی ایجاد شده و یا ارسال پاسخ با شکست مواجه شده است. در صورت مفقود شدن پاسخ، آنگاه ارسال مجدد درخواست ممکن است منجر به انجام دوباره عملیات شود. در عملیات مفروض " مبلغ ۱۰۰۰۰ دلار از حساب بانکی من انتقال بده" ، مشخص است که روش بهتر، دادن گزارش خطاست. اما در عملیات " چه مقدار پول در حسابم باقی مانده" ارسال مجدد درخواست کاملاً بدون اشکال است. عملیاتی را که بتوان آنرا بدون هیچ مشکلی چندین مرتبه تکرار کرد، اصطلاحاً دارای احتمال نشانه گذاری مجدد<sup>۲۲۲</sup> خوانده می شود. از آنجاییکه برخی درخواست ها دارا این ویژگی و برخی دیگر فاقد آن هستند، پس هیچ راه حل واحدی برای توصیه درباره موضوع مفقود شدن پیامها وجود ندارد. در فصل ۸ به طور مفصل راجع به نحوه برخورد با خرابی در ارسال پیام بحث خواهیم کرد.

<sup>222</sup> Idempotent

در روش دیگر، بسیاری از سیستم‌های مشتری-سرور از یک پروتکل اتصال گرای مطمئن استفاده می‌کنند. هرچند این راه حل به دلیل کارآیی نسبتاً پایین در شبکه‌های محلی مناسب نیست، در سیستم‌های حوزه گسترده<sup>223</sup> که ارتباطات ذاتاً غیرمطمئنی دارند، عملکرد خیلی خوبی دارد. عنوان نمونه، تقریباً تمامی پروتکل‌های برنامه کاربردی اینترنت براساس اتصالات قابل وثوق TCP/IP استوار هستند. در این حالت، هرزمانی که مشتری درخواست سرویس نماید، پیش از ارسال درخواست، ابتدا اتصالی را با سرور برقرار می‌نماید. سرور هم برای ارسال پیام پاسخ غالباً از همین اتصال استفاده نموده و سپس اتصال ازبین می‌رود. مشکل این است که ایجاد و ازبین بردن اتصال، خصوصاً در صورت کوچک بودن پیامهای درخواست و پاسخ، تاحدودی هزینه براست.

#### لایه بندی برنامه کاربردی

سالهاست که مدل مشتری-سرور بحث‌ها و مناظره‌های زیادی براه اندخته است. یکی از مهمترین مسائل در همه این سال‌ها ایجاد تمایزی آشکار بین مشتری و سرور بوده است. اما تعجبی ندارد اگر بگوییم که در اغلب موارد هیچ تفاوت آشکاری بین ایندو وجود ندارد. عنوان مثال، سرور یک پایگاه داده ای توزیعی ممکن است دائماً عنوان مشتری عمل کند، به این ترتیب که درخواست‌ها را به سرورهای فایل مختلفی ارسال می‌کند که مسئول پیاده سازی جداول پایگاه داده ای هستند. در چنین حالتی، خود سرور پایگاه داده ای اساساً کاری غیر از پردازش پرس و جوها انجام نمی‌دهد.

با این وجود، از آنجایی که هدف بسیاری از برنامه‌های کاربردی مشتری-سرور پشتیبانی از دسترسی کاربر به پایگاههای داده ای است، بسیاری از کارشناسان براساس شیوه معماری لایه ای که اخیراً مورد بحث قرار گرفت، تفاوت میان سه سطح زیر توصیه کرده اند:

- ۱. لایه رابط/کاربر
- ۲. لایه پردازش
- ۳. لایه داده

سطح کاربر-رابط حاوی تمامی ضروریات برای رابطه مستقیم با کاربر از قبیل مدیریت نمایش است. لایه پردازش حاوی برنامه‌های کاربردی است. سطح داده مدیریت داده‌های واقعی را که روی آنها عمل می‌شود بر عهده دارد.

مشتریان معمولاً سطح رابط کاربر را پیاده سازی می‌کنند. این لایه حاوی برنامه‌هایی است که برای کاربر نهایی امکان تعامل با برنامه‌های کاربردی را فراهم می‌آورد. تفاوت عمده ای بین برنامه‌های پیشرفته رابط کاربر وجود دارد.

---

<sup>223</sup> Wide area

ساده ترین برنامه رابط کاربر چیزی بیش از یک صفحه نمایش براساس کاراکتر نیست. از چنین رابط هایی نوعاً در محیط های کامپیوتری بزرگ<sup>۲۲۴</sup> استفاده می شود. در مواردی که کامپیوتر بزرگ تمامی تعامل ها از جمله با صفحه کلید و با صفحه نمایش را تحت کنترل دارد، دیگر نمی توان از محیط مشتری/سرور سخنی به میان آورد. با این وجود، در بسیاری از موارد ترمینال کاربر برخی از پردازش های محلی جزئی از قبیل انکاس حروف تایپ شده یا پشتیبانی از رابط های فرم ها را انجام می دهد که باید پیش از ارسال ورودی به کامپیوتر اصلی بطور کامل ویرایش شوند.

امروزه، حتی در محیط کامپیوتراهای بزرگ هم شاهد واسطه های کاربر پیشرفته تری هستیم. در اغلب موارد، ماشین مشتری حداقل یک صفحه نمایش گرافیکی ارائه می دهد که در آن از منوهای بالا آوری<sup>۲۲۵</sup> یا پایین آوری<sup>۲۲۶</sup> استفاده شده و بسیاری از کنترل های صفحه به جای صفحه کلید توسط موس انجام می شود. از نمونه های چنین واسطه هایی می توان به واسطه های ویندوز X اشاره کرد که در بسیاری از محیط های UNIX استفاده می شود، همچنین می توان به واسطه های قدیمی تر که در کامپیوتر های شخصی Apple Macintosh و MS-DOS کاربرد داشت اشاره کرد.

واسطه های کاربر امروزی با ایجاد امکان اشتراک برنامه های کاربردی در یک پنجره گرافیکی واحد و استفاده از این پنجره برای تبادل داده ها توسط کاربر میزان کارکرد را تا حد بسیار زیادی افزایش داده اند. بعنوان مثال، برای حذف یک فایل، معمولاً می توان آیکن نماینده آنرا به آیکن سطل آشغال انتقال داد. به همین ترتیب، بسیاری از پردازشگرهای کلمه<sup>۲۲۷</sup> به کاربر امکان می دهد تا با استفاده از موس، متن داخل یک سند مفروض را به موقعیت دیگری انتقال دهند. در فصل ۳ مجدداً در مورد واسطه های کاربر بحث خواهیم کرد.

بسیاری از برنامه های کاربردی مشتری- سرور را می توان بکمک سه بخش تقریباً متفاوت ایجاد نمود: بخشی که کار تعامل با کاربر را در اختیار دارد، بخشی که روی یک پایگاه داده ای یا سیستم فایل عمل می کند، و بخش میانی که غالباً شامل کارکرد اصلی برنامه کاربردی است. این بخش میانی منطقاً در سطح پردازش واقع است. برخلاف واسطه کاربر و پایگاه داده ای، سطح های پردازشی جوانب مشترک زیادی ندارد. بنابراین، برای درک بهتر این لایه چندین نمونه را ذکر خواهیم کرد.

بعنوان اولین نمونه، یک موتور جستجوی اینترنت را درنظر بگیرید. اگر بنرهای متحرک، تصاویر و دیگر زینت های پنجره را در نظر نگیریم، رابط کاربر موتور جستجو بسیار ساده است: کاربر، زنجیره ای از کلمات کلیدی را تایپ کرده و سپس، لیستی از عنوانین صفحات وب در اختیار او گذاشته می شود. پس زمینه<sup>۲۲۸</sup> از پایگاه داده ای عظیمی از صفحات وب تشکیل می شود که از پیش واکشی شده و فهرست

<sup>224</sup> Mainframe

<sup>225</sup> Pop-up

<sup>226</sup> Pull-down

<sup>227</sup> Word processors

<sup>228</sup> Back-end

بندی شده اند. هسته موتور جستجو را برنامه ای تشکیل می دهد که زنجیره کلمات کلیدی کاربر را به یک یا چند پرس و جو در پایگاه داده ای تبدیل می کند. سپس، نتایج را به صورت لیست رتبه بندی کرده و لیست را به مجموعه ای از صفحات HTML ترجمه می کند. در مدل مشتری-서ور، این بخش بازیابی اطلاعات نوعاً در لایه پردازشی قرار داده می شود، همان طور که در شکل ۲-۴ دیده می شود.

**شکل ۲-۴ - سازمان ساده شده یک موتور جستجوی اینترنت به صورت سه لایه** (داخل شکل: رابط کاربر، سطح رابط کاربر، بیان کلمات کلیدی، صفحه HTML حاوی لیست، تولیدکننده HTML تولیدکننده پرس و جو، پرس و جوهای پایگاه داده، لیست رتبه بندی شده حاوی عناوین صفحات، الگوریتم رتبه بندی، سطح پردازشی، عناوین صفحات وب حاوی فوق اطلاعات، پایگاه داده حاوی صفحات وب، لایه داده).

بعنوان مثال دوم، یک سیستم تصمیم یار برای تالار بورس را در نظر بگیرید. این سیستم را هم می توان مشابه موتور جستجو به یک نرم افزار پیش-انتهایی<sup>۲۲۹</sup> اجرا کننده رابط کاربر و یک نرم افزار پس-انتهایی<sup>۲۳۰</sup> جهت دسترسی به پایگاه داده اطلاعات مالی و برنامه های آنالیزی بین ایندو تقسیم کرد. جهت تجزیه و تحلیل اطلاعات مالی ممکن است از روش ها و تکنیکهای پیشرفته آماری و هوش مصنوعی استفاده شود. در برخی موارد، ممکن است حتی لازم باشد که هسته سیستم تصمیم یار مالی روی کامپیوترهای با کارآیی بالا اجرا شود تا بدین ترتیب، گذردهی و پاسخ دهی که کاربران آن انتظار دارند، حاصل شود.

بعنوان آخرین نمونه، یک بسته رومیزی معمول متشکل از یک پردازشگر کلمه<sup>۲۳۱</sup>، یک برنامه کاربردی صفحه گسترده<sup>۲۳۲</sup>، امکانات ارتباطی و غیره را در نظر بگیرید. این مجموعه های "آفیس" غالباً با رابط کاربر مشترک که از استناد ترکیبی پشتیبانی نموده مجتمع می شوند و بر روی فایلهایی از دایرکتوری کاربر عمل می کنند. (در محیط آفیس، چنین دایرکتوری غالباً روی یک سرور فایل دور قرار داده میشود). در این مثال، سطح پردازشی متشکل از مجموعه نسبتاً بزرگی از برنامه هاست که هریک از امکانات پردازشی نسبتاً ساده ای برخوردار هستند.

سطح داده در یک مدل مشتری - سرور شامل برنامه هایی است که از داده های واقعی که برنامه های کاربردی روی آنها عمل می کنند، نگهداری می کنند. یکی از ویژگی های مهم این لایه آن است که داده ها غالباً پایدار<sup>۲۳۳</sup> هستند، یعنی حتی در صورتی هم که هیچ برنامه کاربردی در حال اجرا نباشد، داده جهت استفاده های بعدی در جایی ذخیره خواهد شد. این لایه در ساده ترین شکل خود، متشکل

<sup>229</sup> Front-end

<sup>230</sup> Back-end

<sup>231</sup> Word processor

<sup>232</sup> Spread sheet

<sup>233</sup> Persistent

از یک سیستم فایلی است ، اما معمولتر آن است که از یک پایگاه داده ای تمام عیار استفاده شود. در مدل مشتری-سرور، لایه داده نوعاً در طرف سرور پیاده سازی می شود.

لایه داده علاوه بر ذخیره داده ها، غالباً مسئول حفظ انسجام<sup>۲۳۴</sup> داده ای در تمام برنامه های کاربردی مختلف هم می باشد. در مورد پایگاههای داده ای، حفظ انسجام به این معناست که ابرداده هایی از قبیل شرح جداول، محدودیتهای ورودی و ابرداده های ویژه برنامه کاربردی هم در این لایه ذخیره خواهند شد. بعنوان مثال در امور بانکداری، می خواهیم به محض رسیدن بدھی کارت اعتباری یکی از مشتریان به مقداری مشخصی ، برای او اطلاعیه ای صادر کنیم . این نوع اطلاعات را می توان بوسیله تریگر<sup>۲۳۵</sup> پایگاه داده ای حفظ نمود. این تریگر یک راه انداز<sup>۲۳۶</sup> را در زمان مناسب فعال خواهد ساخت.

در اغلب محیط های کسب و کار گرا، لایه داده بصورت پایگاه داده ای رابطه ای سازمان داده می شود. در اینجا استقلال داده اهمیت کلیدی پیدا خواهد کرد. داده ها ، مستقل از برنامه های کاربردی و به صورتی سازمان داده می شوند که نه تغییرات آن سازمان تأثیری بر برنامه های کاربردی داشته و نه برنامه های کاربردی تأثیری بر سازمان داده ای باقی بگذارند. از آنجایی که لایه پردازش و لایه داده مستقل از یکدیگر تلقی می شوند، بنابراین استفاده از پایگاههای داده ای رابطه ای در مدل مشتری-سرور به ما در تفکیک ایندو لایه بسیار کمک خواهد کرد.

با این وجود، پایگاههای داده ای رابطه ای همواره بهترین گزینه نیستند. یکی از ویژگیهای بارز بسیاری از برنامه های کاربردی این است که روی انواع داده های پیچیده عمل کرده و بنابراین مدلسازی آنها براساس شیء بسیار آسانتر از مدلسازی براساس روابط خواهد بود . از انواع این داده ها می توان به چندضلعی ها و دوایر ساده تا نمایش طرحهای هواییما اشاره کرد که در سیتمهای طراحی بكمک کامپیوتر (CAD) استفاده می شود.

در مواردی که بیان عملیات روی داده از طریق دستکاری شیء راحت تر است، توصیه می شود که لایه داده با استفاده از پایگاه داده ای شیء گرا یا شیء- رابطه پیاده سازی شود. لازم به ذکر است که پایگاههای شیء- رابطه رواج و کاربرد گسترده ای پیدا کرده است، چون این پایگاههای داده ای علاوه بر ارائه مزایای مدل شیء گرا، براساس مدل داده رابطه ای بسیار پراکنده هم استوار هستند.

### **۲۳۷ معماری های چندطبقه ای**

ایده تقسیم بندی به سه سطح منطقی که تا به اینجا مورد بحث قرار گرفت، امکاناتی را برای توزیع فیزیکی برنامه های کاربردی مشتری-سرور در میان چند ماشین مختلف مطرح می کند. در ساده ترین سازمان ، ممکن است فقط با دو نوع ماشین سروکار باشیم:

۱. ماشین مشتری که فقط شامل برنامه های اجراکننده (بخشی از) لایه رابط / کاربر است

<sup>234</sup> Consistency

<sup>235</sup> Trigger

<sup>236</sup> Handler

<sup>237</sup> Multitiered

۲. و ماشین سرور که شامل مابقی برنامه ها ، یعنی برنامه های اجراکننده لایه داده و پردازش است.

در این سازمان ، همه امور توسط سرور هدایت شده و مشتری اساساً چیزی بیش از یک ترمینال غیرهوشمند و احتمالاً دارای یک رابط گرافیکی زیبا نخواهد بود. احتمالات دیگری هم وجود دارند که برخی از معمول ترین آنها رادر ادامه همین بخش مورد بحث قرار خواهیم داد.

یک رویکرد برای سازماندهی مشتری و سرور، توزیع برنامه های لایه های برنامه کاربردی مشروح در بخش قبل در میان ماشینهای مختلف مطابق شکل ۲-۵ است [ همچنین مراجعه شود به (۱۹۹۷): Umar Jing و گروه همکاران ] . بعنوان اولین گام، بین فقط دو نوع ماشین تفاوت قائل می شویم: یکی ماشینهای سرور و دیگری ماشینهای مشتری که باعث ایجاد چیزی به نام معماری دوطبقه ای<sup>۲۳۸</sup> (فیزیکی) میشود:

شکل ۲-۵- انواع سازمانهای سرور- مشتری الف) ( داخل شکل:الف) رابط کاربر، برنامه کاربردی، پایگاه داده، ب) رابط کاربر، برنامه کاربردی، پایگاه داده، ج) ماشین مشتری، رابط کاربر، برنامه کاربردی، پایگاه داده، د) رابط کاربر، برنامه کاربردی، پایگاه داده، ه) رابط کاربر، برنامه کاربردی، پایگاه داده، پایگاه داده).

یکی از روشهای سازماندهی آن است که مطابق شکل الف) ۲-۵، فقط بخش وابسته به ترمینال رابط مشتری روی ماشین مشتری قرار داشته باشد و کنترل برنامه های کاربردی برای ارائه داده های خود از دور انجام شود . یک روش دیگر، قرار دادن تمامی نرم افزار رابط کاربر در طرف مشتری مطابق شکل ب) ۲-۵ است. در چنین مواردی، اساساً برنامه کاربردی به نرم افزار پیش-انتهایی گرافیکی تقسیم می کنیم که از طریق یک پروتکل ویژه برنامه کاربردی، با مابقی برنامه کاربردی ( واقع در سرور) ارتباط برقرار می کند. در این مدل، نرم افزار پیش-انتهایی ( نرم افزار مشتری) هیچ پردازشی بجز آنچه برای ارائه رابط کاربر لازم است انجام نمی دهد.

در ادامه همین مسیر منطقی می توانیم بخشی از برنامه کاربردی را هم مطابق شکل ج) ۲-۵ به نرم افزار پیش-انتهایی انتقال دهیم. یکی از نمونه های کاربرد این روش هنگامی است که در برنامه کاربردی از فرمی استفاده شود که بایستی پیش از پردازش بطور کامل پر شود. پس نرم افزار پیش-انتهایی می تواند صحت و انسجام فرم را بررسی کرده و در صورت لزوم ، با کاربر تعامل نماید. مثال دیگر از سازمان شکل ج) ۲-۵، پردازشگر کلمه ای است که در آن کارکردهای اصلی ویرایشی در طرف مشتری اجرا می شود. این کارکردها روی داده های داخل حافظه نهان محلی یا در حافظه داده پیاده سازی می شوند، در حالی که ابزارهای پشتیبانی پیشرفته ای از قبیل چک کردن املاء و گرامر در طرف سرور اجرا می شوند.

<sup>238</sup> Two-tiered

در بسیاری از محیط های مشتری-سرور، سازمان های نشان داده شده در شکل (۵-۲) و شکل (۵-۲) کاربرد زیاد دارند. از این سازمان ها در مواردی استفاده می شوند که ماشین مشتری، کامپیوتر شخصی یا ایستگاه کاری است که از طریق شبکه به یک سیستم فایل توزیعی یا پایگاه داده ای متصل می باشد. در اینصورت بخش اعظم برنامه کاربردی روی ماشین مشتری اجرا می شود، اما تمامی عملیات روی فایلها و ورودی های پایگاه داده ای به سرور محول می شود. بعنوان مثال، بسیاری از برنامه های کاربردی بانکداری روی ماشین کاربر انتهایی اجرا می شود که در آن کاربر، نقل و انتقالات و اموری از این دست را آماده می کند. پس از پایان کار، برنامه کاربردی با پایگاه داده واقع در سرور بانک ارتباط برقرار کرده و نقل و انتقالات را جهت پردازش بیشتر به آن بالاگذاری<sup>۲۳۹</sup> می کند. شکل (۵-۲) حالتی را نشان می دهد که در آن دیسک محلی مشتری حاوی بخشی از داده هاست. در اینصورت مشتری می تواند مثلاً در حین جستجو در وب، بتدریج حافظه نهان بزرگی از آخرین صفحات وب مشاهده شده را بصورت محلی روی دیسک ایجاد نماید.

اشاره می کنیم که طی چندسال اخیر، تمایل شدیدی نسبت به دوری از پیکربندی های شکل (۵-۲) و شکل (۵-۲) برای مواردی بوجود آمده که نرم افزار مشتری در ماشین های کاربر انتهایی قرار داشته باشد. در این موارد، عمدۀ کار ذخیره سازی و پردازش داده در طرف مشتری انجام می شود. دلیل امر ساده است: هرچند ماشین های مشتری کار زیادی انجام می دهند، ولی مدیریت آنها مشکل زا است. افزایش کارکردی ماشین های مشتری باعث افزایش احتمال خطای نرم افزار طرف مشتری و همچنین وابستگی آن به بستر زیربنایی مشتری (یعنی منابع و سیستم عامل) می شود. از نقطه نظر مدیریت سیستم، داشتن اصطلاحاً مشتریان فربه<sup>۴۰</sup> بهترین گزینه نیست. بلکه بهتر است از مشتریان لاغر<sup>۴۱</sup> سازمان های شکل های (الف) تا (ج) ۲-۵ استفاده شود. البته هزینه این کار احتمالاً استفاده از واسطه های کاربر قدیمی تر و کاهش کارآبی از دیدگاه مشتری خواهد بود.

توجه داشته باشید که این جهت گیری به این معنا نیست که دیگر نیازی به سیستم های توزیعی نخواهیم داشت. بلکه برعکس، راه حل های طرف سرور روز به روز توزیع یافته تر می شوند، زیرا بجائی یک سرور واحد از چندین سرور استفاده می شود که روی ماشین های مختلف اجرا می شوند. بخصوص، در هنگام تمایز بین فقط ماشینهای سرور و مشتری که تا به اینجا انجام شد، این نکته را از قلم انداختیم که ممکن است در بعضی موارد لازم باشد تا مطابق شکل (۶-۲)، سرور بعنوان مشتری ایفای نقش نموده و معماری سه طبقه<sup>۴۲</sup> (فیزیکی) را بوجود آورد.

<sup>239</sup> Upload

<sup>240</sup> Fat clients

<sup>241</sup> Thin clients

<sup>242</sup> Three-tiered

شکل ۲-۶- نمونه ای از مواردیکه سرور بعنوان مشتری عمل می کند) داخل شکل: رابط کاربر(ارائه)، انتظار برای بازگشت، عملیات درخواست، بازگشت نتیجه، انتظار برای داده، سروربرنامه کاربردی، درخواست داده، بازگشت داده، سرور پایگاه داده، زمان).

در این معماری، برنامه هایی که بخشی از سطح پردازش را تشکیل می دهند ممکن است روی یک سرور جداگانه قرار داشته باشند، اما تاحدودی در ماشینهای مشتری و سرور هم توزیع شده باشند. یکی از نمونه های استفاده از معماری سه طبقه در پردازش تراکنش است. همانطور که در فصل ۱ هم گفتیم، فرآیندی جداگانه به نام ناظر پردازش تمامی تراکنش ها بین سرورهای داده احتمالاً مختلف را هماهنگ خواهد کرد.

نمونه دیگر اما بسیار متفاوت که غالباً در معماری های سه طبقه شاهد هستیم ، سازمان و ب سایت ها است. در این حالت، یک سرور وب بعنوان نقطه ورودی به سایت عمل کرده و جهت پردازش واقعی درخواستها، آنها را به سرور برنامه کاربردی عبور می دهد. این سرور برنامه کاربردی هم متقابلاً با سرور پایگاه داده ای تعامل دارد. بعنوان مثال، یک سرور برنامه کاربردی ممکن است مسئول اجرای کد جهت نظارت بر لیست برخی کالاهای موجود در یک مغازه لوازم الکترونیک باشد. بدین منظور، بایستی با پایگاه داده حاوی داده های خام موجودی تعامل نماید. در فصل ۱۲ مجدداً به موضوع سازمان و ب سایت ها باز خواهیم گشت.

## ۲-۲-۲- معماری های غیرمت مرکز

معماری های سرور- مشتری چند طبقه ای یکی از پیامدهای مستقیم تقسیم برنامه های کاربردی به رابط/کاربر، مؤلفه های پردازشی و یک لایه داده ای هستند. طبقات مختلف مستقیماً با سازمان منطقی برنامه های کاربردی مربوط هستند. در بسیاری از محیط های کسب و کاری، پردازش توزیعی بمعنای سازماندهی یک برنامه کاربردی سرور- مشتری بصورت معماری چندطبقه است. ما از این نوع توزیع بنام **توزيع عمودی<sup>۲۴۳</sup>** یاد می کنیم . ویژگی بارز توزیع عمودی آن است که با قرارگیری مؤلفه های منطقاً متفاوت روی ماشینهای مختلف بوجود می آید. این اصطلاح با مفهوم تکه کردن عمودی<sup>۲۴۴</sup> ارتباط دارد که در پایگاه های داده رابطه ای توزیعی بکار برده شده و به این معناست که جداول به صورت ستونی تفکیک و سپس، در میان ماشینهای مختلف توزیع می شوند) ۱۹۹۹ (Oszu and Valdoriez).

مجدداً باید اشاره کرد که از نقطه نظر مدیریت سیستم ، توزیع عمودی می تواند باعث تقسیم شدن منطقی و فیزیکی کارکردها در بین ماشینهای متعددی شود که هریک برای گروه خاصی از کارکردها سفارشی شده است . با این وجود، توزیع عمودی فقط یکی از روشهای سازماندهی برنامه های کاربردی مشتری- سرور است. در معماری های مدرن ، غالباً تکیه بر روی توزیع مشتری ها و سرورها قرار می

<sup>243</sup> Vertical distribution

<sup>244</sup> Vertical fragmentation

گیرد که از آن بانام **توزيع افقی**<sup>۲۴۵</sup> یاد می شود. در این نوع توزیع، یک مشتری یا سرور بایستی به بخش‌های منطبقاً برابر تقسیم شود؛ هر بخش روی سهم و مجموعه کامل داده‌های خود عمل می کند، و به این ترتیب بار متعادل خواهد شد. در این بخش، نگاهی خواهیم داشت به گروهی از معماری‌های نوین سیستم به نام **سیستم‌های نظیر به نظیر**<sup>۲۴۶</sup> که از توزیع افقی پشتیبانی می کنند.

از دیدگاه سطح بالا، فرآیندهای تشکیل دهنده سیستم‌های نظیر به نظیر همگی برابر هستند. به این معنا که کارکردهایی که باید اجرا شوند، بوسیله هریک از فرآیندهای تشکیل دهنده سیستم‌های توزیعی ارائه می شوند. در نتیجه، قسمت اعظم تبادل بین فرآیندها متقاض بوده و هر فرآیند بطور همزمان هم عنوان مشتری و هم سرور عمل خواهد کرد (و بنابراین اصطلاحاً به نام **پیشخدمت**<sup>۲۴۷</sup> هم نامیده می شود).

براساس این رفتار متقاض، مسئله مهم در معماری‌های نظیر به نظیر نحوه سازماندهی فرآیندها در یک شبکه **فوکانی**<sup>۲۴۸</sup> است. و اما شبکه فوکانی شبکه‌ای است که در آن گره‌ها بوسیله فرآیندها شکل گرفته و پیوندها بیانگر کانال‌های ارتباطی ممکن هستند (که معمولاً به توسط اتصالات TCP انجام می شوند). بطور کلی، هیچ فرآیندی نمی تواند مستقیماً با فرآیند اختیاری دیگر ارتباط برقرار کند، بلکه پیامها باید از طریق کانال‌های ارتباطی موجود ارسال شوند. دو نوع شبکه فوکانی وجود دارد: شبکه‌های دارای ساختار و شبکه‌های فاقد ساختار. این دو نوع بطور مفصل در مرجع (۲۰۰۵) Lau و گروه Aberer (۲۰۰۵) همکاران همراه با ذکر مثالهای متعدد مربوطه مورد بررسی قرار گرفته اند. کتاب (۲۰۰۴) و گروه همکاران مرجع خوبی برای انواع معماری‌های ساختار و امکان مقایسه رسمی تر بین انواع مختلف سیستم‌های نظیر به نظیر را فراهم می آورد. بررسی بر اساس دیدگاه توزیع محتوی را در (۲۰۰۴) Androusellis-Theotokis and Spinellis مطالعه کنید.

### معماری‌های نظیر به نظیر دارای ساختار

در معماری‌های نظیر به نظیر دارای ساختار، شبکه فوکانی با استفاده از یک روال مشخص ایجاد می شوند. پرکاربردترین روال تا به امروز سازماندهی فرآیندها با استفاده از **جدول توزیعی درهم (DHT)**<sup>۲۴۹</sup> است. در سیستم‌های براساس جدول توزیعی درهم، به هریک از آیتم‌های داده ای کلید تصادفی از یک فضای بزرگ شناسه ای از قبیل یک شناسه ۱۲۰ یا ۱۶۰ بیتی تخصیص داده می شود. بعلاوه، به گره‌های داخل سیستم هم یک عدد تصادفی از همان فضای شناسه داده می شود. بنابراین مسئله اصلی سیستم‌های براساس DHT، اجرای طرحی مشخص و مؤثر است که بتواند بكمک یک مقیاس سنجش فاصله، شناسه گره منحصر به فردی را به کلید یک آیتم داده ای مفروض،

<sup>245</sup> Horizontal distribution

<sup>246</sup> Peer-to-peer systems

<sup>247</sup> Servent

<sup>248</sup> Overlay network

<sup>249</sup> Distributed hash table

نگاشت نماید (Balakrishnan, ۲۰۰۳). مهم ترین مسئله این است که در هنگام جستجوی یک آیتم داده ای ، آدرس شبکه ای گره مسئول آن آیتم داده ای بازگردانده خواهد شد. در عمل، اینکار توسط مسیریابی درخواست آن آیتم داده ای به گره مربوطه انجام خواهد شد.

بعنوان مثال، در سیستم Chord (۲۰۰۳)، Stocia و گروه همکاران) گره ها منطقاً بصورت یک حلقه سازماندهی شده و آیتم داده ای با کلید  $k$  به گره ای با کوچکترین شناسه  $k \leq id$  نگاشت می شود. این گره را اصطلاحاً جانشین<sup>۲۵۰</sup> کلید  $k$  خوانده و مطابق شکل ۲-۷، بصورت  $succ(k)$  نمایش می دهیم. بنابراین برای جستجوی واقعی یک آیتم داده ای مفروض، لازم است که برنامه کاربردی در حال اجرا بروی گره اختیاری، تابع LOOKUP( $k$ ) را فراخوانی نماید. به این ترتیب، آدرس شبکه ای  $succ(k)$  بازگردانده خواهد شد. در اینجا، برنامه کاربردی می تواند ضمن تماس با گره، کپی از آیتم داده ای مورد نظر را به دست آورد.

## شکل ۲-۷- نگاشت آیتم داده ای روی گره های Chord (داخل شکل: گره واقعی، کلیدهای داده مربوطه).

در حال حاضر برای جستجوی کلید به سراغ الگوریتم های مربوطه نمی رویم و بحث درمورد آنها را به فصل ۵ وجزئیات سیستم های نامگذاری مختلف موکول می کنیم. در عوض، نگاهی خواهیم داشت به نحوه سازماندهی گره ها در یک شبکه فوقانی و یا به بیان دیگر، مدیریت عضویت<sup>۲۵۱</sup>. در ادامه بحث باید بدانیم که جستجوی یک کلید مفروض از سازمان منطقی گره ها در حلقه شکل ۲-۷ تبعیت نمی کند. بلکه هر گره میانبرهایی به دیگر گره ها به نحوی ایجاد می کند که بتواند این دست جستجوها را طی ( $O(\log(N))$  مرحله انجام دهد-  $N$  درواقع تعداد گره های مشارکت کننده در فوقانی است.

حال مجدداً مثال Chord را درنظر می گیریم. هنگامی که گرهی بخواهد به سیستم ملحق شود، کار را با تولید شناسه تصادفی  $id$  آغاز خواهد کرد. توجه داشته باشید که مشروط بر بزرگ بودن کافی فضای شناسه، چنانچه تولید کننده عدد تصادفی از کیفیت خوبی برخوردار باشد، احتمال تولید شناسه ای که در همان زمان به یک گره واقعی تخصیص داده شده باشد، نزدیک به صفر خواهد بود. سپس، گره می تواند به راحتی جستجو را روی  $id$  انجام دهد که اینکار باعث بازگشت آدرس شبکه ای  $succ(id)$  خواهد شد. در اینجا، گره الحاقی می تواند به راحتی با  $succ(id)$  و گره قبل از آن تماس برقرار و خود را در حلقه درج کند. البته، این طرح مستلزم آن است که هر گره اطلاعات را روی گره قبل از خود هم ذخیره کند. درج کردن همچنین باعث خواهد شد تا هر آیتم داده ای که کلید آن با گره  $(id)$  مربوط است، از  $succ(id)$  انتقال داده می شود.

خروج از حلقه هم بسیار ساده است: گره  $(id)$  جانشین و گره قبل از خود را از ترک حلقه آگاه کرده و آیتمهای داده ای خود را به  $succ(id)$  انتقال می دهد.

<sup>250</sup> Successor

<sup>251</sup> Membership Management

در دیگر سیستم های براساس DHT هم از روش‌های مشابه استفاده می شود. بعنوان مثال، شبکه آدرس دهی محتوا<sup>۲۵۲</sup> (CAN) را در نظر بگیرید که در مرجع (۲۰۰۱) Ratnasamy و گروه همکاران شرح داده شده است. در CAN از فضای d بعدی مختصات دکارتی استفاده می شود که بطور کامل در بین تمامی گره های مشارکت کننده در آن سیستم تقسیم بندی شده است. برای ایجاد شکل ذهنی از اینمورد، اجازه دهید فقط حالت دو بعدی را درنظر بگیریم که نمونه ای از آن در شکل ۲-۸ ارائه شده است.

**شکل ۲-۸-الف)** نگاشت آیتم های داده ای روی گره ها در CAN.ب) تقسیم منطقه در هنگام الحق یک گره.

شکلالف) ۲-۸، نحوه تقسیم فضای دوبعدی  $[1, 0] \times [0, 1]$  درین شش گره را نمایش می دهد. هر گره دارای یک منطقه مربوطه بوده و به هریک از آیتمهای داده ای موجود در CAN نقطه منحصر به فردی در این فضا داده می شود. پس از این دیگر مشخص است که کدام گره مسئول آن داده می باشد (آیتم های داده ای واقع روی مرز چند منطقه در نظر نمی گیریم. در این حالت از قانون تخصیص مشخص استفاده می شود).

اگر گره P بخواهد به سیستم CAN ملحق شود، نقطه اختیاری را از فضای مختصاتی انتخاب کرده و سپس گره Q را جستجو خواهد کرد که این نقطه در آن واقع می شود. این جستجو از طریق مسیریابی براساس موقعیت انجام خواهد شد که در پایان همین فصل مورد بحث قرار خواهد گرفت. سپس گره Q منطقه خود را به دونیم تقسیم کرده و همانطور که در شکل ب) ۲-۸ مشاهده می کنید، یک نیمه به گره P تخصیص می یابد. گره ها، گزارشات همسایگان خود یعنی گره های مسئول منطقه مجاور را نگه می دارند. در هنگام تقسیم یک منطقه، گره الحقی P می تواند صرفاً با سؤال از گره Q براحتی همسایگان جدید را شناسایی کند. مشابه Chord، در اینجا هم آیتمهای داده ای که گره P اکنون مسئول آنهاست، از گره Q انتقال داده می شوند.

مسئله خروج در CAN تا حدودی مشکلتر است. فرض کنید که در شکل ۲-۸، گره دارای مختصات (۰،۰،۶)، (۰،۷،۰)، (۰،۹،۰) قصد خروج داشته باشد. منطقه آن به یکی از همسایگانش مثلاً گره واقع در (۰،۰،۹) تخصیص خواهد یافت، اما مشخص است که صرف ادغام گره و ساخت یک مستطیل کافی نخواهد بود. در این حالت، گره واقع در (۰،۹،۰) فقط مراقب آن منطقه بوده و همسایگان قدیمی را از این خبر جدید مطلع خواهد کرد. اینکار باعث کاهش تقارن در فضای مختصات خواهد شد. به همین دلیل یک فرآیند پس زمینه ای متناوباً اقدام به تقسیم بندی مجدد کل فضا خواهد نمود.

**معماری های نظیر به نظیر بدون ساختار**

<sup>252</sup> Content addressable network

در سیستم های نظیر به نظیر بدون ساختار برای ساخت شبکه فوقانی تا حد زیادی بروی الگوریتم های تصادفی تکیه می شود. ایده اصلی این است که هرگره لیستی از همسایه ها را در اختیار داشته باشد؛ اما این لیست باید بصورت کمایش تصادفی ایجاد شود. علاوه فرض می شود که اقلام داده ای به صورت تصادفی روی گره ها قرار می گیرند. در نتیجه، وقتی گرهی اقدام به تعیین مکان یک قلم داده ای خاص می نماید، تنها روش ممکن و مؤثر این است که شبکه را با سیل<sup>۲۵۳</sup> پرس و جو جستجو مواجه سازد (Risson and Moors, ۲۰۰۶). در فصل ۵ مجدداً به موضوع جستجو در شبکه های فوقانی بدون ساختار باز خواهیم گشت و اکنون روی موضوع مدیریت عضویت تمرکز خواهیم کرد.

یکی از اهداف بسیاری از سیستم های نظیر به نظیر بدون ساختار، ساخت شبکه فوقانی نظیر گراف تصادفی<sup>۲۵۴</sup> است. مدل اصلی این است که هر گره حاوی لیستی از  $c$  همسایگان بوده و در حالت ایده آل، هر همسایه نماینده یک گره زنده تصادفاً انتخاب شده از مجموعه فعلی گره ها باشد. لیست همسایگان بنام نمای بخشی<sup>۲۵۵</sup> خوانده می شود. راه های مختلفی برای ساخت نمای بخشی وجود دارد. در مرجع (Jelasity ۲۰۰۴، ۲۰۰۵) و گروه همکاران چارچوبی برای در بر گیری الگوریتم های مختلف برای ساخت شبکه فوقانی ارائه داده اند تا امکان ارزیابی و مقایسه هریک فراهم آید. در این چارچوب، فرض می شود که گره ها بصورتی منظم مدخل ها را از نمای بخشی خود تبادل می نمایند. هریک از مدخل ها گره دیگری را در شبکه شناسایی نموده و برای بیان قدمت آن گره در منبع، از عددی بعنوان سن استفاده می شود. مطابق شکل ۲-۹، در این رابطه از دو نخ<sup>۲۵۶</sup> استفاده می شود.

## شکل ۲-۹- (الف) مراحل انجام شده بوسیله نخ فعال. (ب) مراحل انجام شده بوسیله نخ غیرفعال

نخ فعال برای اولین بار اقدام به برقراری ارتباط با گره دیگر می کند. این گره از نمای بخشی فعلی آن انتخاب می شود. با این فرض که ورودی ها باید به نظیر انتخابی هل<sup>۲۵۷</sup> داده شوند، کار را با ایجاد یک بافر حاوی  $c/2+1$  مدخل از جمله ورودی شناسنده خود ادامه خواهد داد. مابقی مدخل ها از نمای بخشی فعلی گرفته می شوند.

چنانچه گره در حالت کشیدن<sup>۲۵۸</sup> باشد،منتظر پاسخی از جانب نظیر انتخابی باقی خواهد ماند. در این فاصله زمانی، ما هم بوسیله نخ غیرفعال شکل (ب) ۲-۹، یک بافر ساخته ایم. فعالیت های این نخ بسیار شبیه نخ فعال نظیر خود است.

<sup>253</sup> Flood

<sup>254</sup> Random Graph

<sup>255</sup> Partial view

<sup>256</sup> Thread

<sup>257</sup> Push

<sup>258</sup> Pull

نکته مهم، ساخت نمای بخشی جدید است. این نما برای هر دو نظیر تماس گرفته شده و آغازین دقیقاً حاوی  $C$  مدخل خواهد بود و بخشی از آن از بافر دریافتی بدست خواهد آمد. بطور خلاصه، دو روش برای ساخت نمای بخشی وجود دارد. در حالت اول، دو گره ممکن است اقدام به دورریختن مدخل هایی کنند که قبلاً برای هم ارسال کرده بودند. اینکار باعث خواهد شد تا بخشی از نماهای اصلی آنها با هم مبادله شود. روش دوم دورریختن حداکثری مدخل های قدیمی است. در مجموع باید گفت که ایندو روش در واقع مکمل هم هستند<sup>259</sup> برای مطالعه جزئیات مراجعه شود به (۵۰۰) Jelarsity و گروه همکاران<sup>۱</sup>. می توان نتیجه گرفت که بسیاری از پروتکل های مدیریت عضویت برای شبکه های فوکانی بدون ساختار از این چارچوب تبعیت می کنند. چندین مشاهده جالب وجود دارد.

اولاً، فرض کنیم که وقتی گرهی تصمیم به الحق می گیرد، با گره اختیاری دیگر که احتمالاً از روی لیستی از نقاط دسترسی معروف انتخاب خواهد شد، تماس برقرار می نماید. این نقطه دسترسی صرفاً عضو دائمی از شبکه فوکانی است و همچنین بسیار دسترسی پذیر می باشد. در این حالت می توان نتیجه گرفت که پروتکل هایی که فقط از روش هل دادن یا فقط از روش کشیدن استفاده می کنند، ممکن است براحتی شبکه های فوکانی غیرمتصل ایجاد کنند. به بیان دیگر، گروه های گره ها ایزوله شده و دیگر توان دسترسی به همه گره های موجود در شبکه را نخواهند داشت. ناگفته پیداست که این ویژگی مطلوب نبوده و به همین دلیل معقولانه تر این است که گره ها بتوانند واقعاً مدخل ها را با هم تبادل نمایند.

ثانیاً، بشرطی که گره ها بصورتی منظم اقدام به تبادل نماهای بخشی نمایند، عملیات خروج از شبکه بسیار آسان خواهد شد. در این حالت، هر گره می تواند به راحتی و بدون اطلاع دادن به گره های دیگر خارج شود. در واقع، هنگامیکه گره مفروض  $P$  یکی از همسایگان ظاهری خود مثلاً  $Q$  را انتخاب می کند و می فهمد که  $Q$  دیگر پاسخ نخواهد داد، مدخل را از نمای بخشی خود حذف و نظیر دیگری را انتخاب خواهد کرد. بنابراین گره در هنگام ساخت نمای بخشی جدید، از سیاست دورریزی حداکثر ممکنه مدخل ها تبعیت خواهد کرد و گره های خارج شده به سرعت فراموش می شوند. به بیان دیگر، مدخل های ارجاع شده به گره های خروجی بصورت اتوماتیک و سریعاً از نماهای بخشی حذف خواهند شد.

با این وجود، تبعیت از این راهکار هزینه هایی هم در پی دارد. به ازای گره  $P$ ، مجموعه ای از گره ها را فرض کنید که در نمای بخشی خود مدخلی دارد که به  $P$  ارجاع می کند. این ویژگی از نظر فنی درجه مدخلیت<sup>۲۵۹</sup> گره نامیده می شود. با افزایش زاویه درونی گره  $P$ ، احتمال برقراری تماس توسط گره های دیگر با گره  $P$  هم افزایش خواهد یافت. به بیان دیگر، این خطر وجود دارد که  $P$  تبدیل به گره مورد علاقه شده و بارکاری زیاد وضعیت نامتعادلی را در گره ایجاد کند. دورریختن مدخل های قدیمی

---

<sup>259</sup> Indegree

به طور سیستماتیک باعث ترغیب گره ها به گره های دارای درجه مدخلیت بیشتر می شود. انواع دیگری هم از مقایسه ها وجود دارند که خوانندگان مشتاق را به مطالعه آنها در مرجع (۲۰۰۵) Jelarsity و گروه همکاران دعوت می کنیم.

### مدیریت توپولوژی شبکه های فوکانی

هرچند ممکن است به نظر برسد که سیستم های نظیر به نظیر دارای ساختار و بدون ساختار کاملاً از یکدیگر مستقل هستند، در عمل ضرورتاً اینطور نیست [ مراجعه شود به (۲۰۰۵) Castro و گروه همکاران]. یکی از مهمترین مشاهدات این است که با تبادل و انتخاب آگاهانه ورودی ها از نماهای بخشی، امکان ساخت و نگهداری توپولوژی های مشخص شبکه های فوکانی فراهم می آید. مدیریت این توپولوژی با اتخاذ روش دولایه ای مطابق شکل ۲-۱۰ امکان پذیر می شود.

شکل ۲-۱۰- روش دولایه ای برای ساخت و نگهداری از توپولوژی های مشخص فوکانی با استفاده از روش های سیستم های نظیر به نظیر بدون ساختار(داخل شکل: فوکانی دارای ساختار، پروتکل مربوط به فوکانی مشخص، پیوند به دیگر گره های دارای توپولوژی مشخص، نظیر تصادفی، فوکانی تصادفی، پروتکل مربوط به نمای تصادفی، پیوند به دیگر گره های تصادفاً انتخاب شده).

پایین ترین لایه، سیستم نظیر به نظیر بدون ساختار را تشکیل می دهد که گره های موجود در آن به منظور کسب یک گراف تصادفی دقیق، متناوباً مدخل های نماهای بخشی خود را مبادله می کنند. در این مورد دقت به این مطلب اشاره دارد که نمای بخشی باید با مدخل های رجوع کننده به گره های زنده تصادفاً انتخاب شده پر شود.

پایین ترین لایه نمای بخشی خود را به لایه بالاتر عبور داده که در آنجا انتخاب دیگری روی مدخل ها انجام می شود. اینکار منجر به تهیه لیست ثانویه ای از همسایگان متناظر با توپولوژی مورد نظر خواهد شد. (۲۰۰۵) Jelarsity and Babaoglu پیشنهاد داده اند که با استفاده از مفهومی به نام تابع رتبه بندی<sup>۲۶۰</sup>، گره ها بر حسب ضابطه ای مرتبط با آن گره مفروض مرتب شوند . وظیفه یک تابع ساده رتبه بندی آن است که با افزایش فاصله از گره مفروض  $P$  مجموعه ای از گره ها را مرتب کند. در این حالت، چنانچه پایین ترین لایه همچنان به عبوردادن گره های تصادفاً انتخاب شده خود ادامه دهد، گره  $P$  تدریجاً لیستی از نزدیکترین همسایگان خود ایجاد خواهد کرد.

برای درک بهتر یک ساختار شبکه منطقی با ابعاد  $N \times N$  را تصور کنید که در آن یک گره روی هریک از نقاط شبکه قرار گرفته باشد. هر گره باید حاوی لیستی از تعداد  $C$  نزدیکترین همسایگان باشد. فاصله بین گره واقع در  $(a_2, a_1)$  و  $(b_2, b_1)$ ، به صورت  $d_2 + d_1 - b_i$  تعریف شده و در آن  $/$

<sup>260</sup> Ranking function

۲-۹ اقدام به اجرای چنانچه پایین ترین لایه متنابه<sup>۲۶۱</sup> و مطابق شکل  $d = \min(N - a_i, b_i)$  پروتکل نماید، توپولوژی حاصله به یک توروس<sup>۲۶۲</sup> شبیه شکل ۲-۱۱ نزدیک خواهد شد.

شکل ۲-۱۱- تولید یک شبکه فوقانی مشخص با استفاده از یک سیستم نظیر به نظیر دولایه ای بدون ساختار [اقتباس از Jelasity and Babaoglu (۲۰۰۵)]

البته از توابع رتبه بندی کاملاً متفاوتی می توان استفاده کرد که در آن میان ، توابع مربوط به کسب مجاورت معنایی<sup>۲۶۳</sup> به اقلام داده ای که در گره نظیر ذخیره شده اند، از بقیه جالبتر است . این مجاورت، امکان ایجاد شبکه های فوقانی معنایی را فراهم می کند که امکان ساخت الگوریتم های جستجوی بسیار مؤثر در سیستم های نظیر به نظیر بدون ساختار را می دهد. در فصل ۵ در حین بحث درمورد نامگذاری براساس ویژگی مجدداً به این سیستم ها بازخواهیم گشت.

### ابرنظیرها

قابل ذکر است که در سیستم های نظیر به نظیر بدون ساختار، با رشد شبکه، مکان یابی آیتمهای داده ای مرتبط هم مشکل تر خواهد شد. دلیل این مشکل در مقیاس پذیری آشکار است: چون هیچ روش کاملاً مشخص برای مسیریابی درخواست جستجو به یک قلم داده ای مفروض وجود ندارد و اساساً تنها روشی که یک گره مفروض می تواند بدان متمسک شود، ارسال سیل درخواست است. روش‌های مختلفی برای مانع سازی در برابر این سیل وجود دارد که در فصل ۵ با آنها اشاره خواهیم کرد. اما روش جایگزین برای بسیاری از سیستم های نظیر به نظیر، استفاده از گره های ویژه ای است که ایندکسی از اقلام داده ای را در خود نگه دارد.

نمونه های دیگری از مواردی که بهتر است از ماهیت متقارن سیستم های نظیر به نظیر صرفنظر کرد وجود دارد. مثلاً تشریک مساعی گره هایی را در نظر بگیرید که منابعی را به یکدیگر ارائه می کنند. بعنوان مثال، در یک شبکه تحويل محتوا<sup>۲۶۴</sup> (CDN) با تشریک مساعی، ممکن است گره ها فضاهای ذخیره سازی برای کپی های صفحات وب ارائه کرده تا به این ترتیب، مشتریان وب بتوانند به صفحات دسترسی پیدا کرده و در نتیجه امکان دسترسی سریع برای آنها فراهم شود. در این حالت، ممکن است لازم باشد که گره مفروض  $P$  در بخش خاصی از شبکه به دنبال منابع مورد نظر خود بگردد. در این حالت، استفاده از واسطه<sup>۲۶۵</sup> ای برای جمع آوری کاربری منبع برای تعدادی از گره های واقع در مجاورت یکدیگر، امکان انتخاب سریع گره دارای منابع کافی را فراهم می آورد.

گره هایی از قبیل آنها که ایندکس را نگه می دارند یا بعنوان واسطه عمل می کنند، غالباً بنام ابرنظیر خوانده میشوند. همانطور که از نام آنها هم برمند آید، ابرنظیرها هم غالباً در شبکه های نظیر به نظیر

<sup>261</sup> Torus

<sup>262</sup> Semantic proximity

<sup>263</sup> Superpeers

<sup>264</sup> Content delivery network

<sup>265</sup> Broker

سازماندهی شده و منجر به تولید سازمان سلسله مراتبی مشابه چیزی می شوند که در مرجع (۲۰۰۳) Yang and Garcia- Molina تشریح شده است. یکی از انواع ساده این سازمانها در شکل ۲-۱۲ نمایش داده شده است. در این سازمان، هر یک از نظیرهای عادی بعنوان مشتری به یک ابرنظیر متصل شده است . تمامی ارتباطات از / به نظیر عادی از طریق ابرنظیر مربوط به همان نظیر انجام خواهد شد.

**شکل ۲-۱۲- سازمان سلسله مراتبی گره ها در یک شبکه ابرنظیر (داخل شکل: نظیر عادی، ابرنظیر، شبکه ابرنظیر).**

در بسیاری از موارد، رابطه مشتری - ابرنظیر ثابت است؛ یعنی هر وقت که یک نظیر عادی به شبکه ملحق می شود، به یکی از ابرنظیرها متصل شده و تا زمان خروج از شبکه این اتصال را حفظ خواهد کرد. قاعdetتاً انتظار می رود که ابرنظیرها فرآیندهایی با عمر طولانی و دسترس پذیری بالا باشند. برای جبران رفتار بعضاً ناپایدار یک ابرنظیر، می توان از طرحهای پشتیبان از قبیل جفت کردن هر ابرنظیر با ابرنظیردیگر و الزام مشتریان برای اتصال به هردو استفاده نمود.

داشتن ارتباط ثابت با یک ابرنظیر ممکن است همواره بهترین راه حل نباشد. بعنوان مثال، در مورد شبکه های اشتراک فایل ، شاید گزینه بهتر برای مشتری اتصال به ابرنظیری است که حاوی ایندکس فایل هایی باشد که مشتری غالباً به آنها تمایل دارد. در این حالت، احتمال اطلاع ابرنظیر از محل فایل خاص مورد جستجوی مشتری بالا خواهد بود. (۲۰۰۵) Garbacki و گروه همکاران طرح نسبتاً ساده ای ارائه کرده اند که می توان در آن، در صورتیکه مشتری ها ابرنظیرهای بهتری را برای مشارکت پیدا کنند، رابطه مشتری- ابرنظیر را تغییر داد. مشخصاً ابرنظیری که نتیجه عملیات جستجو را بازمی گرداند، نسبت به دیگر ابرنظیرها ارجحیت دارد.

همانطور که تا به اینجا دیدیم، شبکه های نظیر به نظیر ابزار انعطاف پذیری را برای پیوستن به شبکه یا خروج از شبکه برای گره ها ارائه می دهند. با این وجود، شبکه های ابرنظیر مشکل تازه ای را مطرح می کنند که عبارتست از نحوه انتخاب گره هایی که شایسته تبدیل شدن به ابرنظیر هستند. این مسأله ارتباط نزدیکی با مسئله انتخاب هماهنگ کننده دارد که در فصل ۶ در حین بحث در مورد انتخاب ابرنظیرها در یک شبکه نظیر به نظیر راجع به آن صحبت خواهیم کرد.

### **۲-۲-۳- معماری های هیبریدی (پیوندی)**

تا به اینجا روی معماری های مشتری- سرور و تعدادی از معماری های نظیر به نظیر تمرکز کرده ایم. در بسیاری از سیستم های توزیعی، ویژگی های معماری با هم ترکیب می شوند، همچنان که در شبکه های ابرنظیری مطرح کردیم. در این بخش نگاهی خواهیم داشت به برخی از انواع خاص سیستم های توزیعی که در آنها راه حلهای مشتری- سرور با معماری های غیرمت مرکز ترکیب شده است.

**سیستم های سرور لبه ای**

<sup>266</sup> Edge-server systems

یکی از مهمترین انواع سیستم‌های توزیعی که براساس معماری هیبریدی سازماندهی شده، سیستم‌های سرورلبه‌ای است. این سیستمها روی اینترنت در جاهایی پیاده شده اند که سرورها "در لبه شبکه" قرار می‌گیرند. این لبه براساس مرز بین شبکه‌های شرکتی و اینترنت واقعی تشکیل می‌شود که مثلاً بوسیله فراهم کنندگان سرویس اینترنت<sup>۲۶۷</sup> (ISP) ارائه می‌شود. به همین ترتیب، هنگامیکه کاربران نهایی خانگی از طریق ISP خود به اینترنت متصل می‌شوند، می‌توان تصور کرد که ISP در لبه اینترنت واقع شده و در نهایت سازمانی مشابه شکل ۲-۱۳ بوجود خواهد آمد.

**شکل ۲-۱۳- نمای اینترنت بصورت مجموعه‌ای متشکل از سرورهای لبه‌ای (داخل شکل: مشتری، فراهم کننده محتوا، سرور لبه‌ای، هسته اینترنت، شبکه شرکتی).**

کاربران نهایی، یا بطور کلی مشتریان، بوسیله سرور لبه‌ای به اینترنت متصل می‌شوند. وظیفه اصلی سرور لبه‌ای خدمات رسانی به محتوا، احتمالاً پس از اعمال کارکردهای فیلترینگ و تبدیل کد است. جالب‌تر اینکه از مجموعه‌ای از سرورهای لبه‌ای می‌توان برای بهینه سازی محتوا و توزیع برنامه کاربردی استفاده نمود. در مدل اصلی این گونه است که برای یک سازمان خاص یک سرور لبه‌ای می‌تواند به عنوان سرور مبدأ عمل کند که تمام محتوا از آن نشأت می‌گیرد. این سرور هم می‌تواند از دیگر سرورهای لبه‌ای جهت کپی برداری از صفحات وب و نظایر آن استفاده کند (Rabinovich, ۲۰۰۲؛ Leff, ۲۰۰۴؛ Nayate, ۲۰۰۴؛ and Spatscheck, ۲۰۰۴). در فصل ۱۲ در مبحث مربوط به راه حل‌های براساس وب مجدداً به این موضوع برخواهیم گشت.

#### سیستم‌های توزیعی با تشریک مساعی<sup>۲۶۸</sup>

ساختارهای هیبریدی علی الخصوص در سیستم‌های توزیعی با تشریک مساعی پیاده سازی شده اند. مسئله اصلی بسیاری از این سیستم‌ها اولین راه اندازی است که بدین منظور غالباً یک طرح مشتری-سرور متعارف اجرا شده است. پس از الحق یک گره مفروض به سیستم، این گره می‌تواند جهت تشریک مساعی از طرحی کاملاً غیرمتمرکز استفاده کند.

برای درک موضوع ابتدا نگاهی خواهیم داشت به سیستم اشتراک فایل BitTorrent (Cohen, ۲۰۰۳). BitTorrent یک سیستم دانلود فایل نظریه نظری است که عملکرد اصلی آن در شکل ۲-۱۴ ارائه شده است. ایده اصلی این است که وقتی کاربر نهایی به دنبال فایل خاصی می‌گردد، تکه هایی از فایل را آنقدر از کاربران دیگر دانلود کند که این تکه‌ها بتوانند با کنار هم قرار گرفتن، یک فایل کامل را ایجاد کنند. یکی از مهمترین اهداف طراحی این سیستم‌ها تضمین تشریک مساعی بوده است. در غالب سیستم‌های اشتراک فایل، بخش اعظمی از شرکت کنندگان صرفاً فایلها را دانلود می‌کنند، و در بقیه موارد نقش آنها چیزی نزدیک به صفر است (Yang, ۲۰۰۵؛ Saroiu, ۲۰۰۳؛ و Adar and Huberman, ۲۰۰۰). بنابراین، یک فایل را فقط هنگامی می‌توان دانلود

<sup>267</sup> Internet service provider

<sup>268</sup> Collaborative distributed systems

کرد که مشتری در حال دانلود، در حال تهیه محتوا برای فرد دیگری باشد. کمی بعدتر مجدداً به این رفتار "تلافی جویانه" بازخواهیم گشت.

شکل ۲-۱۴ - اصل کلی نحوه عملکرد BitTorrent [اقتباس با کسب اجازه از (۲۰۰۴) Pouwelse و گروه همکاران] (داخل شکل: تعداد k گره از N گره، گره مشتری، جستجوی F، صفحه وب BitTorrent سرور وب، ارجاع به سرور فایل، فایل torrent، سرور F، سرور فایل، ارجاع به ردیاب، لیست گره های ذخیره کننده F در ردیاب، گره ۱، گره ۲، گره N).

جهت دانلود فایل، کاربر باید به دایرکتوری جهانی دسترسی داشته باشد که فقط یکی از چندین وب سایت محدود شناخته شده و معروف است. چنین دایرکتوری حاوی ارجاعاتی به چیزی بنام فایلهای torrent است. یک فایل torrent حاوی اطلاعات لازم برای دانلود یک فایل خاص است. مشخصاً به چیزی به نام ردیاب<sup>۲۶۹</sup> اشاره دارد که در واقع سروری است برای نگهداری گزارش دقیقی از گره های فعالی که حاوی (تکه های) فایل درخواست شده باشند. گره فعال گرهی است که در حال دانلود کردن فایل دیگری باشد. ناگفته پیداست که ردیابهای مختلفی ممکن است وجود داشته باشند، اما غالباً فقط یک ردیاب برای هر فایل (یا مجموعه فایل ها) موجود است.

پس از آنکه گره ها از محل مناسب برای دانلود تکه ها اطلاع پیدا کردند، گره در حال دانلود فعال خواهد شد. در این صورت گره مجبور می شود به دیگر گره ها کمک کند، مثلاً با ارائه تکه هایی از فایلی که او در حال دانلود و دیگران فاقد آن هستند. این اجبار ناشی از یک قانون بسیار ساده است: چنانچه گره P مطلع شود که گره Q در حال دانلود کردن بیش از آن چیزی است که آپلود می کند، P ممکن است تصمیم بگیرد که از سرعت خود در ارسال داده ها به Q بکاهد. مادامی که P چیزی برای دانلود کردن از Q داشته باشد، این طرح به خوبی عمل خواهد کرد. به همین دلیل، غالباً مرجع هایی به بسیاری از گره های دیگر در اختیار گره ها قرار می گیرد تا در وضعیت بهتری برای معامله قرار گیرند. ناگفته پیداست که BitTorrent صورت ترکیبی از راه حل های متمرکز و غیرمتمرکز است. بنابراین نقطه ضعف این دست سیستمهای در واقع همین ردیابها هستند.

عنوان نمونه دیگر، شبکه توزیع محتوا با تشریک مساعی Pierre and Van Globule (Steen ۲۰۰۶) را درنظر بگیرید. معماری Globule تا حد زیادی مشابه معماری سرور لبه ای است که قبلاً بحث شد. در این حالت، بجای سرور لبه ای، کاربران نهایی (و همچنین سازمانها) بطور داوطلبانه سرورهای وب تقویت شده ای را ایجاد می کنند که قادر به تشریک مساعی در کپی کردن صفحات وب هستند. در ساده ترین شکل، چنین سروری دارای مؤلفه های زیر است:

۱. مؤلفه ای که می تواند درخواستهای مشتری را به طرف سرورهای دیگر هدایت کند.
۲. مؤلفه ای برای آنالیز کردن الگوهای دسترسی.

<sup>269</sup> Tracker

### ۳. مؤلفه ای برای مدیریت کپی برداری از صفحات وب.

سرور ایجاد شده بوسیله کاربر مفروض آلیس سرور وبی است که معمولاً ترافیک مربوط به وب سایت آلیس را هدایت کرده و سرور مبدأ آن سایت نامیده می شود. این سرور با سرورهای دیگر، مثلاً سرور ایجادشده بوسیله باب تشریک مساعی کرده و میزبان صفحات سایت باب می شود. از این نقطه نظر، Globule یک سیستم توزیعی غیر مرکز محسوب می شود. درخواستهای مربوط به وب سایت آلیس درابتدا به سرور او ارسال می شوند و ممکن است از آنجا به یکی دیگر از سرورها هدایت شود. از هدایت مجدد توزیعی هم پشتیبانی می شود.

با این وجود، Globule یک مؤلفه مرکز دیگری هم به شکل **واسطه<sup>۲۷۰</sup>** دارد. این واسطه مسئول ثبت سرورها و شناساندن آنها به دیگر سرورها می باشد. نحوه ارتباط سرورها با واسطه کاملاً مشابه چیزی است که از سیستم های مشتری- سرور انتظار می رود. جهت ایجاد قابلیت دسترسی، واسطه را می توان کپی نمود؛ اما همانطور که در ادامه همین کتاب خواهیم دید، این نوع کپی برداری به میزان گسترده ای جهت تضمین قابلیت اطمینان محاسبات سرور- مشتری مورد استفاده قرار می گیرد.

### ۲-۳- مقایسه معماری ها با میان افزار

با بررسی مسائل معماري که تا به اینجا مورد بحث قرار گرفت، این پرسش مطرح می شود که میان افزار در این بین چه جایگاهی دارد. همانطور که در فصل ۱ و شکل ۱-۱ هم مشاهده کردیم، میان افزار لایه ای را بین برنامه های کاربردی و سکو های توزیع شده ایجاد می کند. یکی از مهم ترین اهداف اینکار، ایجاد درجه ای از شفافیت توزیعی است که بتوان بكمک آن توزیع داده، توزیع پردازش و توزیع کنترل را از برنامه های کاربردی مخفی کرد.

اما غالباً در عمل مشاهده می شود که سیستم های میان افزاری از شیوه معماري خاصی تبعیت می کنند. بعنوان مثال، بسیاری از راه حلهای میان افزاری از قبیل CORBA (OMG، ۲۰۰۴) از شیوه معماري براساس شیء استفاده می کنند. برخی دیگر، از جمله TIB/Rendezvous (TIBCO، ۲۰۰۵) برای ایجاد میان افزار از شیوه معماري براساس رویداد بهره می گیرند. در فصل های بعد، به برخی از نمونه های شیوه های معماري اشاره خواهیم کرد.

مزیت میان افزارهایی که براساس شیوه معماري خاصی مدل شده اند، این است که بعضاً باعث تسهیل طراحی برنامه های کاربردی می شوند. اما یکی از نقاط ضعف آشکار آن این است که میان افزار مذکور ممکن است برای برنامه کاربردی بهینه نباشد. بعنوان مثال، CORBA در ابتدا فقط اشیائی را ارائه می داد که توسط مشتری های از راه دور قابل فراخوانی بود. بعد ها احساس شد که استفاده صرف از این نوع تعامل بسیار محدود کننده خواهد بود و به همین دلیل، دیگر الگوهای تعامل از قبیل پیا م دهی

<sup>270</sup> Broker

هم به آن افزوده شد. ناگفته پیداست که افزودن ویژگی های جدید می تواند باعث افزایش راه حل های میان افرازی شود.

علاوه، هرچند هدف از میان افزار ایجاد شفافیت توزیعی است، معمولاً تصور می شود که بهتر است برخی راه حل ها با شرایط برنامه های کاربردی سازگاری پیدا کنند. یک راه حل آن است که نسخه های متعددی از یک سیستم میان افزاری تهیه کرده و هر نسخه را بحسب گروه خاصی از برنامه های کاربردی سفارشی کنیم. روشی معمولاً مفیدتر این است که سیستم های میان افزاری را به نحوی ایجاد کنیم که بحسب برنامه کاربردی مورد نظر به راحتی قابل پیکربندی، سازگاری و تغییر باشند. در نتیجه، امروزه سیستم هایی در حال ایجاد هستند که در آنها تفکیک بیشتر و مشخص تری بین سیاست ها و مکانیزم ها منظور شده باشد. از همین رو، مکانیزم های متعددی جهت بهبود رفتار میان افزارها بوجود آمده اند (Sadjadi and McKinley, ۲۰۰۳).

اجازه دهید در ادامه نگاهی داشته باشیم به برخی از متدائل ترین روش ها در این زمینه.

### ۱-۲-۳- راه گیرها<sup>۲۷۱</sup>

از نظر مفهومی، راه گیر در واقع ساختار نرم افزاری است که جریان عادی کنترل راشکسته و به کد) ویژه برنامه کاربردی) دیگر اجازه اجرا می دهد. عمومی کردن راه گیرها مستلزم تلاش اجرایی زیادی است که توسط (Schmidt ۲۰۰۰) و گروه همکاران) مورد بررسی قرار گرفته است . بطور قطع نمی توان گفت که در چنین مواردی ارجحیت با عمومیت است یا قابلیت اجرای محدود و ساده بودن. همچنین، در بسیاری از موارد داشتن امکانات راه گیری محدود باعث بهبود مدیریت نرم افزار و سیستم توزیعی خواهد شد.

برای استحکام مطلب، موضوع راه گیری را به صورتی در نظر بگیرید که در بسیاری از سیتمهای توزیعی براساس شیء مورد پشتیبانی قرار می گیرد. ایده اصلی ساده است: شیء  $A$  قادر به فراخوانی متدهای متعلق به شیء  $B$  است، اما  $B$  روی ماشینی غیراز  $A$  قرار دارد. همانطور که در بخش های بعدی همین کتاب خواهیم گفت، این فراخوانی شیء از راه دور طی روش سه مرحله ای زیر انجام خواهد شد:

۱. رابط محلی<sup>۲۷۲</sup> دقیقاً مانند آنچه توسط شیء  $B$  ارائه می شود به شیء  $A$  ارائه می شود. هم متدهای موجود در آن رابط را فراخوانی می کند.
  ۲. فراخوانی توسط  $A$  به فراخوانی شیء عمومی تبدیل می شود که از طریق رابط فراخوانی شیء عمومی ارائه شده توسط میان افزار موجود در ماشین  $A$ ، امکان پذیر می شود.
  ۳. نهایتاً، فراخوانی شیء عمومی تبدیل به پیامی می شود که توسط رابط شبکه ای سطح انتقال ارائه شده بوسیله سیستم عامل محلی  $A$  ارسال می شود.
- شمای کلی از این مراحل در شکل ۲-۱۵ نمایش داده است.

<sup>271</sup> Interceptors

<sup>272</sup> Local interface

شکل ۲-۱۵- استفاده از راه گیرها جهت انجام فراخوانی های شیء از راه دور( داخل شکل: فراخوانی مسدود شده، راه گیر سطح درخواست، راه گیر سطح پیام، برنامه کاربردی مشتری، برنامه کاربردی مجازی، فراخوانی مسدود نشده، میان افزار شیء، سیستم عامل محلی، به سمت شیء (B).

پس از مرحله اول، فراخوانی `B.do_something(value)` تبدیل به فراخوانی عمومی از قبیل `invoke(B,&do_something,value)` با ارجاع به متند `B` و پارامترهای همراه فراخوانی می شود . حال تصور کنید که شیء `B` کپی برداری شود. در اینصورت ، همه کپی ها باید فراخوانی شوند. رعایت این نکته مورد روشنی است که راه گیرها می توانند کمک کنند. کافی است راه گیر سطح درخواست اقدام به فراخوانی `invoke(B,&do_something,value)` برای هر یک از کپی ها می نماید. ظرفات کار در اینجاست که نه نیاز است که شیء `A` از کپی شدن `B` اطلاع داشته باشد و نه اینکه میان افزار شیء مؤلفه های ویژه ای برای کار با این فراخوانی کپی برداری شده در اختیار داشته باشد . فقط راه گیر سطح درخواست که ممکن است به میان افزار/اضافه شود، باید از کپی شدن `B` مطلع باشد.

بعنوان آخرین نکته باید گفت که فراخوانی یک شیء از راه دور باستی از طریق شبکه ارسال شود. در عمل، این ویژگی لزوم فراخوانی رابط پیام دهی را مطرح می کند که بوسیله سیستم عامل محلی ارائه می شود. در این سطح ، راه گیر سطح پیام ممکن است در انتقال فراخوانی به شیء هدف کمک کند. بعنوان مثال، تصور کنید که پارامتر `value`، واقعاً متناظر با آرایه بزرگی از داده باشد. در این حالت، عاقلانه تر این است که داده را به قطعات کوچکتری تقسیم نمود تا بعداً در مقصد مجدد سرهم بندی شوند. این دست تجزیه کردنها ممکن است باعث ارتقاء میزان کارآیی یا قابلیت اطمینان شود. در این مورد هم لازم نیست که میان افزار از این تجزیه اطلاع پیدا کند؛ راه گیر سطح پایین تر به صورتی شفاف با همکاری سیستم عامل محلی اقدام به هدایت ارتباطات خواهد کرد.

## ۲-۳- رویکردهای عمومی به نرم افزار تطبیقی<sup>۲۷۳</sup>

در واقع می توان گفت که راه گیرها ابزاری جهت تطبیق میان افزار ارائه می کنند. لزوم تطبیق ریشه در این واقعیت دارد که محیط اجرای برنامه های کاربردی توزیعی دائماً دستخوش تغییر می شود. از جمله این تغییرات می توان به جابجایی، تفاوت زیاد کیفیت خدمات شبکه ها، خرابی سخت افزار، تخلیه باتری و بسیاری دیگر اشاره کرد. وظیفه واکنش در برابر این تغییرات بجای برنامه های کاربردی، بر عهده میان افزار قرار داده شده است.

این دست تأثیرات شدید بسیاری از طراحان نرم افزار را به فکر ساخت نرم افزار تطبیقی انداخته است. اما موفقیت نرم افزار تطبیقی هیچگاه به اندازه ای نبوده که از آن انتظار می رفت. از آنجایی که بسیاری از محققان و سازندگان نرم افزار آنرا یکی از مهم ترین جنبه های سیستم های توزیعی امروزی می دانند،

<sup>273</sup> Adaptive software

در ادامه نگاه مختصری به آن خواهیم داشت . در مرجع (۲۰۰۴) McKinley و گروه همکاران سه تکنیک اساسی در مورد تطبیق نرم افزار مطرح شده است.

#### ۱. تفکیک موارد

<sup>۲۷۴</sup> ۲. انعکاس محاسباتی

۳. طراحی براساس مؤلفه

تفکیک موارد با روش متعارف پیمانه ای ساختن سیستم ها مرتبط می باشد، یعنی تفکیک بخشهايی که کارکردها را پیاده سازی می کنند از سایر بخش های مسئول انجام امور دیگر (که به نام فوق کارکرد<sup>۲۷۵</sup> خوانده می شود) از قبیل قابلیت اطمینان، کارآیی، امنیت وغیره. می توان استدلال کرد که ایجاد میان افزار برای برنامه های کاربردی توزیعی تا حدود زیادی با کار با موضوع فوق کارکرد آن هم بصورتی مستقل از برنامه های کاربردی مرتبط است . مشکل اصلی این است که نمی توان از طریق پیمانه ای ساختن، این فوق کارکردها را از هم تفکیک نمود. بعنوان مثال، صرف قرار دادن امنیت در یک پیمانه جداگانه کار نخواهد کرد. به همین ترتیب، نمی توان تصور کرد که تحمل خرابی را در یک جعبه جداگانه قرار داد و بعنوان یک سرویس مستقل ارائه نمود. تفکیک و سپس به هم دوختن این موارد مرتبط در یک سیستم ( توزیعی) مفروض از مهمترین نکاتی است که در ایجاد نرم افزار جنبه گرا<sup>۲۷۶</sup> (۲۰۰۵) و گروه همکاران Filman را از مدنظر قرار می گیرد. با این وجود، جنبه گرایی هنوز بطور موفقیت آمیزی در توسعه سیستم های توزیعی پردازمانه اعمال نشده است و بنابراین تا تحقق این امر هنوز راه زیادی در پیش داریم.

انعکاس محاسباتی اشاره دارد به توانایی یک برنامه مفروض در نظارت برخود و ،در صورت لزوم، انطباق رفتار خود(۲۰۰۲) Kon و گروه همکاران). موضوع انعکاس در برخی از زبان های برنامه نویسی از جمله Java لحاظ شده و امکان گسترش ای را برای اصلاحات زمان اجرا در اختیار قرار می دهد. بعلاوه، برخی از سیستم های میان افزاری امکان اتخاذ روشاهی انعکاسی را بوجود می آورند. با این وجود، درست مانند حالت جهت یابی جنبه، میان افزار انعکاسی باید بتواند خود را بعنوان یک ابزار قوی در مدیریت پیچیدگی سیستم های توزیعی پردازمانه اثبات کند. همانطور که در مرجع (۲۰۰۴) Blair و گروه همکاران هم مطرح شده، اعمال انعکاس در دامنه گسترش برنامه های کاربردی هنوز بطور کامل محقق نشده است .

نهایتاً، طراحی براساس مؤلفه، از تطبیق به طریق ترکیب پشتیبانی می کند. هر سیستم مفروض را می توان یا به صورت ایستا در زمان طراحی پیکربندی کرد و یا به صورت پویا در زمان اجرا. مورد پویا نیازمند پشتیبانی برای پیوند دیرهنگام<sup>۲۷۷</sup> می باشد. این روش به صورت موفقیت آمیزی هم در محیط

<sup>274</sup> Computational reflection

<sup>275</sup> Extra functionality

<sup>276</sup> Aspect-oriented

<sup>277</sup> Late binding

های زبان برنامه نویسی اعمال شده و هم در سیستم‌های عاملی که بتوان در آنها پیمانه‌ها را به دلخواه بارکرد و تخلیه نمود. در حال حاضر، تحقیقات مفیدی در زمینه ایجاد امکان انتخاب اتوماتیک بهترین پیاده‌سازی مؤلفه در حین زمان اجرا در دست انجام است (Yellin, ۲۰۰۳). اما در اینجا هم فرآیند مربوط به سیستم‌های توزیعی پیچیده است. این مشکل خصوصاً هنگامی رخ می‌نماید که بدانیم تعویض یک مؤلفه چه تأثیری بر دیگر مؤلفه‌ها خواهد داشت. در بسیاری از موارد، مؤلفه‌ها استقلال کمتری از آنچه تصور می‌شود دارند.

### ۳-۳-۲- بحث

معماری نرم افزارهای سیستم‌های توزیعی، خصوصاً آنهایی که عنوان میان افزار تلقی می‌شوند، حجمی و پیچیده است. در اکثر موارد، این حجم و پیچیدگی ناشی از لزوم عمومی بودن و آنهم ناشی از لزوم تأمین شفافیت توزیعی است. در عین حال، برنامه‌های کاربردی نیازمندی‌های فوق کارکردی ویژه‌ای دارند که با هدف دستیابی کامل به این شفافیت در تضاد است. این تضاد نیازمندی‌ها در عمومیت بخشیدن و ویژه گرایی منجر به ارائه راه حل‌های میان افزاری بسیار انعطاف پذیر شده است. بهای چنین دستاوردی پیچیده تر شدن است. به عنوان مثال، Zhang and Jacobsen (2004) گزارش داده اند که فقط طی چهار سال اول، ابعاد یکی از محصولات نرم افزاری ۵۰٪ رشد داشته است. این در حالی است که تعداد کل فایلهای آن محصول طی همین دوره سه برابر شده اند. مشخص است که تشویقی برای ادامه این مسیر وجود ندارد.

باتوجه به اینکه تقریباً تمامی سیستم‌های بزرگ نرم افزاری امروزه بایستی در محیط شبکه‌ای شده اجرا شوند، باید از خود بپرسیم که آیا پیچیدگی سیستم‌های توزیعی صرفاً یک ویژگی ذاتی ناشی از تلاش برای ایجاد شفافیت توزیعی است. البته، مسائلی از قبیل باز بودن<sup>۲۷۸</sup> هم به این اندازه حائز اهمیت است، اما لزوم انعطاف پذیری تاکنون هرگز به اندازه برای میان افزارها مطرح نبوده است.

Coyler (۲۰۰۳) و گروه همکاران استدلال می‌کنند که باید تأکید بیشتری روی سادگی، یعنی اتخاذ روشی ساده‌تر برای ساخت میان افزار بوسیله مؤلفه‌ها، و استقلال برنامه کاربردی صورت پذیرد. این که کدام یک از روش‌های فوق الذکر راه حل مورد نظر است محل بحث است. بخصوص که هیچ یک از روش‌های پیشنهادی تا کنون نه مورد پذیرش گسترده واقع شده اند و نه بطور موفقیت آمیزی در سیستم‌های پردازمنه مورد استفاده قرار گرفته اند.

فرضیه زیربنایی آن است که ما نیاز به نرم افزار تطبیقی داریم؛ به این معنا که نرم افزار بایستی همگام با تغییر محیط قادر به تغییر باشد. با این وجود، باید از خود پرسید که آیا تطبیق با تغییرات محیطی دلیل

<sup>278</sup> Openness

خوبی برای پذیرش تغییر نرم افزار خواهد بود. به نظر می رسد که سخت افزار معیوب، حملات امنیتی، تخلیه انرژی و مواردی از این دست همگی تغییرات محیطی هستند که می توانند (و بهتر است) بوسیله نرم افزار پیش بینی شوند.

مستدل ترین ، و قطعاً معتبرترین، بحث در حمایت از نرم افزار تطبیقی این است که امکان توقف فعالیت بسیاری از سیستم های توزیعی وجود ندارد. همین محدودیت لزوم ارائه راه حلهایی را برای تعویض و ارتقاء مؤلفه ها در زمان اجرا مطرح می کند. اما هنوز مشخص نیست که کدامیک از راه حلهای پیشنهادی فوق بهترین شیوه برای حل این دست مشکلات مربوط به نگهداری می باشد. آنچه باقی می ماند این است که سیستم های توزیعی باید بتوانند، مثلًا با تغییر سیاستهای مربوط به تخصیص منابع ، در برابر تغییرات محیط از خود واکنش نشان دهند. تمامی مؤلفه های نرم افزاری لازم جهت ایجاد چنین تطبیقی هم اکنون موجود می باشد. در واقع این الگوریتم های واقع در مؤلفه ها هستند که نحوه رفتار برای تغییر مقادیر را دیکته می کنند. مهم این است که اجزه دهیم این رفتار واکنشی بدون دخالت انسانی انجام شود. مشاهده شده که این رویکرد در صورتی عملکرد بهتری خواهد داشت که مثلًا در حین تصمیم گیری درمورد محل قرار گرفتن مؤلفه ها ، راجع به سازمان فیزیکی آنها بحث می کنیم. راجع به این نوع مسائل معماري سیستم در ادامه بحث خواهیم کرد.

#### ۴-۲- خود مدیریتی در سیستم های توزیعی

سیستم های توزیعی- و بخصوص میان افزار مربوط به آنها- بایستی راه حلهای عمومی برای مقاومت در برابر ویژگی های نامطلوب ذاتی شبکه سازی ارائه دهند به طوری که بتوانند از تعداد زیادی برنامه های کاربردی پشتیبانی نمایند. از طرف دیگر، اغلب برنامه های کاربردی هم عملًا در پی کسب شفافیت توزیعی کامل نیستند و این امر باعث پشتیبانی از راه حل های ویژه برنامه کاربردی می شود. قبلًا استدلال کرده ایم که به همین دلیل، سیستم های توزیعی باید تطبیقی باشند ؛ اما این ضرورت هنگامی است که تطبیق رفتار اجرایی آنها مورد نظر است و نه مؤلفه های نرم افزاری تشکیل دهنده آنها.

هنگامی که ضرورت تطبیق اتوماتیک مطرح می شود ، نوعی کنش متقابل قوی بین معماري های سیستم و معماري های نرم افزار مشاهده می شود. از طرف دیگر، باید مؤلفه های سیستم توزیعی را به نحوی سازماندهی کرد که امکان نظارت و تنظیم فراهم آید. در عین حال، باید در مورد محل مناسب برای اجرای فرآیندهای مجری این انطباق هم تصمیم گیری شود.

در این بخش، نگاه دقیقتراخواهیم داشت به سازماندهی سیستم‌های توزیعی بصورت سیستم‌های سطح بالای کنترل بازخورد<sup>۲۷۹</sup> تا امکان تطبیق اتوماتیک با تغییرات فراهم شود. این پدیده به نام محاسبه اتوماتیک (Kephart, ۲۰۰۳) یا سیستم‌های خود-ستاره<sup>۲۸۰</sup> (Babaoglu, ۲۰۰۵) و گروه همکاران) هم خوانده می‌شود. نامگذاری اخیر بیانگر تنوع روش‌های کسب تطبیق اتوماتیک است، از قبیل خود مدیریتی، خود ترمیمی، خود پیکربندی، خود بهینه سازی و غیره. در ادامه بحث از عنوان سیستم‌های مدیریتی برای اشاره به گونه‌های مختلف این سیستم‌ها استفاده خواهیم کرد.

#### ۱-۴-۲-۴-۱- مدل کنترل بازخورد

دیدگاه‌های مختلفی در مورد سیستم‌های خود مدیریتی وجود دارد، اما نقطه اشتراک (آشکار یا ضمنی) در اغلب آنها این است که تطبیق بوسیله یک یا چند حلقه کنترل بازخورد انجام می‌شود. برهمناس، سیستم‌هایی که بوسیله این نوع حلقه‌ها سازماندهی می‌شوند، به نام سیستم‌های کنترل بازخورد نامیده می‌شوند. سال‌هاست که کنترل بازخورد در حوزه‌های مختلف مهندسی مورد استفاده قرار گرفته و بنيانهای ریاضی آنهم تدریجیاً در سیستم‌های محاسباتی نفوذ پیدا کرده‌اند (Diao, ۲۰۰۵ و گروه همکاران؛ ۲۰۰۴، Helleerstein و گروه همکاران). در سیستم‌های خود مدیریتی، مسائل معماري در نگاه اول جالبترین موضوع بنظر می‌رسند. ایده زیربنایی این سازمان که در شکل ۲-۱۶ مشاهده می‌کنید، بسیار ساده است.

شکل ۲-۱۶- سازمان منطقی یک سیستم کنترل بازخورد (داخل شکل: پارامترهای غیرقابل کنترل (اختلال، نویز)، پیکربندی اولیه، اصلاحات، هسته سیستم توزیعی، خروجی مشاهده شده، اقدامات تنظیمی، ورودی مرجع، برآورد مقیاس، راه اندازهای تنظیم، آنالیز، خروجی اندازه گیری شده).

هسته سیستم کنترل بازخورد از مؤلفه‌هایی تشکیل می‌شود که نیازمند مدیریت هستند. فرض می‌شود که این مؤلفه‌ها از طریق پارامترهای ورودی قابل کنترل رانده می‌شوند، در حالیکه ممکن است رفتار آنها تحت تأثیر هرنوع ورودی غیرقابل کنترل - که به نام ورودی اختلال<sup>۲۸۱</sup> یا نویز هم خوانده می‌شود - قرارگیرد. هرچند اختلال غالباً ناشی از محیط اجرای سیستم توزیعی است، ممکن است در برخی موارد تقابل پیش‌بینی نشده مابین مؤلفه‌ها هم باعث بروز رفتار غیرمنتظره ای شود.

حلقه کنترل بازخورد اساساً از سه المان تشکیل شده است: اولاً، خود سیستم باید تحت نظرارت قرار گیرد، به همین دلیل لازم است که جنبه‌های مختلف سیستم سنجش شوند. در بسیاری از موارد، صحبت از سنجش رفتار آسانتر از عمل به آن است. بعنوان مثال، تأخیرهای مسیر رفت و برگشت در اینترنت ممکن است بی‌رویه تغییر کرده و وابسته به این باشد که دقیقاً چه چیزی اندازه گیری شده

<sup>279</sup> Feedback control

<sup>280</sup> Self-star systems

<sup>281</sup> Disturbance

است. در چنین مواردی ، برآورده دقیق تأخیر بسیار پیچیده تر می شود. این مسئله خصوصاً هنگامی پیچیده تر می شود که گره مفروض  $A$  بخواهد بدون ایجاد مزاحمت برای دو گره کاملاً متفاوت دیگر مثلاً  $B$  و  $C$ ، تأخیر بین آنها را برآورد کند. به دلایلی از همین قبیل، حلقه های کنترل بازخورد غالباً شامل یک مؤلفه برآورده مقیاس<sup>۲۸۲</sup> منطقی هستند.

بخش دیگر حلقه کنترل بازخورد به آنالیز اندازه گیری ها و مقایسه آنها با مقادیر مرجع می پردازد. این مؤلفه آنالیز بازخورد در واقع قلب حلقه کنترل بوده و حاوی الگوریتمهایی برای تصمیم گیری در مورد تطبیق های احتمالی است.

گروه آخر مؤلفه ها شامل مکانیزم های مختلفی برای تأثیرگذاری مستقیم بر رفتار سیستم است که از آن جمله می توان به جاگذاری کپی ها، تغییر اولویت های جداول زمان بندی، جابجایی سرویس های انتقال داده ها جهت ایجاد دسترسی، هدایت مجدد درخواستها به سمت سرورهای مختلف و غیره اشاره کرد. مؤلفه آنالیز باید از این مکانیزم ها و تأثیر (انتظار رفته) از آنها بر رفتار سیستم مطلع باشد. به همین دلیل اقدام به راه اندازی یک یا چند مکانیزم خواهد کرد تا بعداً بتواند تأثیر را مشاهده کند.

یکی از مشاهدات جالب این است که حلقه کنترل بازخورد در مدیریت دستی سیستمهای قابل کاربرد است . تفاوت عمدی این است که بجای مؤلفه آنالیز از مدیران انسانی استفاده می شود. با این وجود، جهت مدیریت مناسب سیستم های توزیعی، این مدیران بایستی تجهیزات ناظارتی و همچنین مکانیزم های کنترل رفتار سیستم را در اختیار داشته باشند. مشخص است که آنالیز صحیح داده های اندازه گیری شده و راه اندازی اقدامات صحیح و بجا، توسعه سیستم های خود مدیریتی را مشکل می سازد. لازم به ذکر است که شکل ۲-۱۶ بیانگر سازمان منطقی یک سیستم خود مدیریتی بوده و به همین دلیل، مشابه مورد بحث ما درباره معماری های نرم افزار است. اما سازمان فیزیکی ممکن است بسیار متفاوت تر باشد. بعنوان مثال، ممکن است مؤلفه آنالیزی بطور کامل در سرتاسر سیستم توزیع شده باشد. همچنین، اندازه گیری کارآئی معمولاً در هر ماشینی که جزء سیستم توزیعی محسوب می شود انجام می گیرد. در ادامه نگاهی خواهیم داشت به چند نمونه ملموس در مورد نحوه نظارت، آنالیز، و اصلاح سیستم های توزیعی بصورتی خودکار. با مطالعه این مثال ها به تفاوت بین سازمان فیزیکی و منطقی نیز توجه خواهیم کرد.

#### ۲-۴-۲-مثال: نظارت بر سیستمهای به کمک **Astrolabe**

بعنوان اولین نمونه، نگاهی خواهیم داشت به **Astrolabe** (Van Renesse، ۲۰۰۳) و گروه همکاران) که در واقع سیستمی برای پشتیبانی از نظارت عمومی بر سیستم های توزیعی بسیار بزرگ است . در مورد سیستم های خود مدیریتی، **Astrolabe** بایستی بعنوان ابزار عمومی برای نظارت بر رفتار

<sup>282</sup> Metric estimation component

سیستمها مدنظر قرار گیرد. از خروجی آن می‌توان جهت تغذیه مؤلفه آنالیز برای تصمیم‌گیری درمورد اقدامات اصلاحی استفاده نمود.

Astrolabe مجموعه بزرگی از میزبانها را بصورت سلسله مراتبی از ناحیه‌ها سازماندهی می‌کند. ناحیه‌های واقع در پایین ترین سطح فقط شامل یک میزبان هستند، که بعداً به صورت گروهی در ناحیه‌هایی با اندازه افزایش یافته گروه بندی می‌شوند. ناحیه واقع در بالاترین سطح تمامی میزبانها را تحت پوشش قرار می‌دهد. هر یک از میزبانان، یک فرآیند Astrolabe بنام agent را اجرا می‌کند که مسئولیت جمع آوری اطلاعات روی ناحیه‌هایی که شامل این میزبان هستند را دارد. همچنین agent با دیگر agent‌ها هم ارتباط برقرار می‌کند تا بدین ترتیب، بتواند اطلاعات منطقه را در سرتاسر سیستم انتشار دهد.

هر میزبان شامل مجموعه‌ای از ویژگی‌ها برای جمع آوری اطلاعات محلی است. بعنوان مثال، یک میزبان مفروض گزارش فایل‌های مشخص شده‌ای را که ذخیره می‌کند، کاربرد منابع سیستم، و از این قبیل را نگهداری کند. فقط ویژگی‌های مستقیماً بیان شده بوسیله میزبانها، یعنی در پایین ترین سطح سلسله مراتب، قابل نوشتگری هستند. بعلاوه، هر ناحیه می‌تواند شامل مجموعه‌ای از ویژگی‌ها باشد، اما مقادیر این ویژگی‌ها بر حسب مقادیر ناحیه‌های سطح پایین‌تر محاسبه می‌شوند.

مطابق شکل ۲-۱۷، مثال ساده‌ای را درنظر بگیرید که در آن سه میزبان A و B و C در یک ناحیه گروه بندی شده‌اند. هریک از ماشینها، گزارش آدرس IP خود، بار CPU، حافظه آزاد موجود و تعداد فرآیندهای فعال را نگه می‌دارد. هریک از این ویژگی‌ها را می‌توان با استفاده از اطلاعات محلی مربوط به هریک از میزبانها مستقیماً نوشت. در سطح ناحیه، فقط اطلاعات متراکمی از قبیل میانگین بار CPU یا میانگین تعداد فرآیندهای فعال را می‌توان جمع آوری نمود.

شکل ۲-۱۷- جمع آوری داده و تراکم اطلاعات در **Astrolabe** (داخل شکل: میانگین بار، میانگین اعضاء، میانگین فرآیندها، ماشین A، ماشین B، ماشین C، آدرس IP، بار، اعضاء، فرآیندها).

شکل ۲-۱۷ نحوه مشاهده اطلاعات گردآوری شده بوسیله هر یک از ماشینها بصورت گزارش در پایگاه داده‌ای و همچنین الحق آنها به یکدیگر جهت تشکیل یک رابطه (جدول) را نمایش می‌دهد. این شکل در واقع بیانگر روش مشاهده تمامی داده‌های جمع آوری شده بوسیله Astrolabe است. با این وجود، اطلاعات به ازای هر ناحیه فقط براساس گزارشات اصلی که در اختیار میزبانان قرار دارد، قابل محاسبه است.

اطلاعات متراکم بوسیله توابع تراکم قابل برنامه‌ریزی بددست می‌آید که بسیار مشابه توابع موجود در زبان پایگاه داده‌ای رابطه‌ای SQL هستند. بعنوان مثال، با این فرض که اطلاعات میزبان شکل ۲-۱۷ در جدول محلی به نام hostinfo نگهداری می‌شود، با انجام یک جستجوی SQL ساده به صورت زیر می‌توان میانگین تعداد فرآیندهای مربوط به ناحیه حاوی ماشینهای BA، C را محاسبه نمود:

`SELECT AVG( procs) AS avg_procs FROM hostinfo`

با چند بهبود جزئی در SQL ، به راحتی می توان پرس و جوهای اطلاعاتی بیشتری را فرمول بندی کرد.

پرس و جوهایی از این دست دائمًا توسط هر یک از agent های درحال اجرا روی میزبان ها ارزیابی می شوند. مشخص است که این ارزیابی فقط در صورتی امکان پذیر خواهد بود که اطلاعات ناحیه به تمامی گره های تشکیل دهنده Astrolabe پخش شده باشد. تا به اینجا agent در حال اجرا برروی یک میزبان مسئول محاسبه بخش هایی از جداول ناحیه های مربوط به خود می باشد. در حالی که گزارشاتی که میزبان مسئولیت محاسباتی در قبال آن ندارد، بعضاً از طریق یک روال تبادلی ساده اما مؤثر بنام **شايعه پراكنی**<sup>۲۸۳</sup> به آن ارسال می شود. پروتکل های شایعه پراكنی در فصل ۴ بحث می شوند. یک agent نتایج محاسبه شده را به agent های دیگر انتقال می دهد.

نتیجه تبادل اطلاعات آن است که نهایتاً ، تمامی agent هایی که برای کمک به کسب برخی اطلاعات متراکم مورد نیاز هستند، ( مشروط براینکه هیچ تغییری در آن زمان انجام نشود) نتایج واحدی را مشاهده خواهند کرد.

### ۳-۴-۲- مثال: تمایز بین راهکارهای کپی برداری در Globule

حال بیایید نگاهی داشته باشیم به یک شبکه توزیع محتوا بر اساس تشریک مساعی به نام Globule (Pierre and Van Steen، ۲۰۰۶)، بر قرارگیری سرورهای کاربر نهایی در اینترنت و مشارکت این سرورها در بهینه سازی عملکرد از طریق کپی برداری از صفحات وب تکیه می کند. بدین منظور، هریک از سرورهای مبدأ (یعنی سرور مسئول انجام به هنگام سازی های یک وب سایت خاص)، گزارش الگوهای دسترسی را براساس دسترسی به هر صفحه نگهداری می کند. الگوهای دسترسی بصورت عملیات نوشتمن و خواندن برای یک صفحه مفروض بیان شده و هر عمل بوسیله سرور مبدأ خاص آن صفحه مهرزمانی خورده و ثبت می شود.

در ساده ترین شکل Globule فرض می کند که اینترنت به صورت یک سیستم سرورلبه ای (قبل از توضیح داده شده است) قابل رؤیت است. مطابق شکل ۲-۱۸، فرض می شود که درخواستها همیشه از طریق یک سرور لبه ای مناسب قابل عبوردادن هستند. این مدل ساده به سرور مبدأ امکان می دهد تا در صورت وجود کپی روی یک سرور لبه ای خاص، از تمامی اتفاقات مطلع شود. هرچند کاهش فاصله کپی از مشتریان باعث تأخیر درک شده بوسیله مشتری می شود، در عین حال ممکن است جهت حفظ انسجام نسخه کپی با نسخه اصلی، موجب افزایش ترافیک بین سرور اصلی و آن سرور لبه ای شود.

<sup>283</sup> Gossiping

## شکل ۲-۱۸- مدل سرورلبه ای مفروض در Globule (مشتری، سرور مبدأ، سرور کپی، شبکه شرکتی، اینترنت مقیم، مشتری).

سرور مبدأ پس از دریافت درخواست در مورد یک صفحه خاص، آدرس IP را از محل مبدأ درخواست ثبت کرده و با استفاده از سرویس اینترنتی WHOIS ، اقدام به جستجوی شبکه شرکتی یا ISP مربوط به آن درخواست می نماید (Deutsch و گروه همکاران). سپس سرور مبدأ به دنبال نزدیکترین سرور کپی موجود می گردد که بتواند بعنوان سرور لبه ای آن مشتری عمل کند. در آخر هم تأخیر تا آن سرور را به همراه پنهانی باند حداکثری محاسبه خواهد کرد. در ساده ترین شکل Globule فرض می شود که تأخیر بین سرور کپی و ماشین کاربر در حال درخواست قابل اغماض بوده، و همچنین پنهانی باند بین این دو بسیار زیاد است.

پس از جمع آوری درخواستهای کافی برای یک صفحه ، سرور مبدأ اقدام به انجام آنالیز ساده "چه می شد اگر" میکند. این آنالیز در قالب ارزیابی سیاست های مختلف کپی برداری می کند. یک سیاست تعیین می کند که صفحه کجا باشد، و نحوه نگهداری انسجام آن صفحه چگونه باشد. هریک از سیاست های کپی برداری هزینه ای درپی دارد که بصورتتابع خطی ساده زیر قابل بیان است:

$$cost = (w_1 \times m_1) + (w_2 \times m_2) + \dots + (w_n \times m_n)$$

که در آن  $m_k$  مقیاس عملکرد و  $w_k$  بیانگر میزان اهمیت آن مقیاس است. از نمونه مقیاس های عملکردی می توان به تأخیر جمع شده بین مشتری و سرور کپی در هنگام بازگشت دادن کپی های صفحات وب، مجموع پنهانی باند مصرف شده بین سرور مبدأ و سرور کپی جهت حفظ انسجام کپی، و تعداد کپی های (مجاز) قدیمی که به مشتری بازگشت داده شده اند، اشاره کرد (Pierre، ۲۰۰۲ و گروه همکاران).

بعنوان مثال، فرض کنید که  $d_c$  تأخیر بین زمان صدور درخواست توسط مشتری  $C$  و بازگشت صفحه از بهترین سرور کپی باشد. توجه داشته باشید که بهترین سرور کپی براساس سیاست کپی برداری تعیین می شود. فرض کنیم که  $m_1$  بیانگر تأخیر جمع شده طی یک دوره زمانی خاص ، یعنی  $\sum d_c = m_1$  باشد . اگر سرور مبدأ بخواهد تأخیر درک شده بوسیله مشتری را بهینه نماید، باید مقدار عددی نسبتاً بزرگی را برای  $w_1$  انتخاب کند. در نتیجه، فقط آنسته از سیاستهایی که واقعاً  $m_1$  را حداقلی می کنند، هزینه نسبتاً ناچیزی را ببار خواهد آورد .

در Globule ، سرور مبدأ به کمک شبیه سازی براساس مسیر<sup>۲۸۴</sup> ، مرتبأ اقدام به ارزیابی چند ده سیاست کپی برداری برای هریک از صفحات جداگانه وب خواهد کرد. از همین شبیه سازی ها بهترین سیاست انتخاب و سپس اجرا می شود. ممکن است اینطور بنظر برسد که کپی های جدید در سرورهای لبه ای مختلفی نصب شده و یا این که از روش متفاوتی برای منسجم نگه داشتن کپی ها استفاده می

<sup>284</sup> Trace-driven simulation

شود. جمع آوری مسیر<sup>۲۸۵</sup>‌ها، ارزیابی سیاستهای کپی برداری، و اعمال سیاست انتخاب شده همگی بصورت اتوماتیک انجام خواهد شد.

چند نکته ظریف دیگر هم وجود دارند که باید مورد بحث قرار گیرد. در درجه اول، هنوز مشخص نیست که چه تعداد درخواست باید پیش از آغاز ارزیابی سیاست جاری جمع آوری شود. برای توضیح، فرض کنید که در زمان  $T_i$ ، سرور مبدأ سیاست  $p$  را برای دروه زمانی بعدی یعنی تا  $T_{i+1}$  انتخاب می کند. این انتخاب براساس مجموعه درخواستهایی صورت می پذیرد که قبلًا بین زمان  $T_i$ ،  $T_{i-1}$  صادر شده بودند. البته، بعداً سرور ممکن است با بازگشت به زمان گذشته  $T_{i+1}$ ، به این نتیجه برسد که بهتر بود براساس درخواستهای واقعی که در فاصله زمانی  $T_i$ ،  $T_{i+1}$  صادر شده بودند، سیاست<sup>\*</sup>  $p$  را انتخاب می کرد.

می توان نتیجه گرفت که درصد پیش بینی های خطابستگی به طول سری درخواستهایی دارد (که اصطلاحاً طول مسیر نامیده می شود) که جهت پیش بینی و انتخاب سیاست بعدی مورد استفاده قرار می گیرد. این وابستگی در شکل ۲-۱۹ ارائه شده است. همانطور که از شکل هم برمی آید، در صورت ناکافی بودن طول مسیر، خطای پیش بینی بهترین سیاست افزایش خواهد یافت. عنوان دلیل باید گفت که برای انجام یک ارزیابی مناسب، درخواست های کافی باید اجرا شوند. با این وجود، درصورت استفاده از تعداد درخواستهای بسیار زیاد، خطاب هم افزایش خواهد یافت. دلیل این است که افزایش بیش از حد طول مسیر باعث بازیابی تعداد زیادی تغییرات در الگوهای دسترسی شده و این امر هم متقابلاً باعث مشکل شدن انتخاب بهترین سیاست و حتی غیرممکن شدن آن خواهد شد. برای درک این پدیده معروف فرض کنید که بخواهیم با بررسی اوضاع جوی صدسال اخیر، هوای فردا را پیش بینی کنیم.

مسلماً اینکار بی فایده خواهد بود و بررسی سابقه چند روز اخیر نتیجه بهتری ایجاد خواهد کرد. یافتن طول بهینه مسیر بصورت اتوماتیک هم امکان پذیر است. ارائه راه حل مناسب برای این مشکل را به عنوان تمرین ارائه شده است.

#### ۴-۲-۴-مثال: مدیریت تعمیر اتوماتیک مؤلفه در Jade

وقتی از مجموعه خوشة ای<sup>۲۸۶</sup> از کامپیوتراهای مجهز به سرورهای پیشرفته نگهداری می کنید، حل مشکلات مدیریتی تبدیل به نوعی ضرورت می شود. یک روش برای سرورهای ساخته شده براساس رویکرد مؤلفه-پایه<sup>۲۸۷</sup>، کشف خرابی مؤلفه ها و تعویض اتوماتیک آنهاست. در سیستم Jade از این رویکرد پیروی می کند (۲۰۰۵، Bouchenak و گروه همکاران). در این بخش مختصرًا راجع به آن بحث خواهیم کرد.

<sup>285</sup> Trace

<sup>286</sup> Cluster

<sup>287</sup> Component-based

براساس مدل مؤلفه Fractal ساخته شده است و شامل یک پیاده سازی Java از یک چارچوب برای افروden و حذف مؤلفه ها در زمان اجرا است (۲۰۰۴، Bruneton و گروه همکاران). هر مؤلفه داخل Fractal ممکن است دو رابط<sup>۲۸۸</sup> داشته باشد: یک رابط سرور که جهت فراخوانی متدهای که توسط آن مؤلفه پیاده سازی شده اند، و یک رابط مشتری که توسط مؤلفه برای فراخوانی مؤلفه های دیگر استفاده می شود. مؤلفه ها توسط واسطه های پیوندی<sup>۲۸۹</sup> بیدیگر متصل می شوند. عنوان مثال، رابط مشتری مؤلفه  $C_1$  را می توان به رابط سرور مؤلفه  $C_2$  پیوند داد. عمل اولیه یوند به این معناست که فراخوانی رابط مشتری مستقیماً منجر به فراخوانی رابط پیوند خورده به سرور می شود. در حالت پیوند ترکیبی فراخوانی ممکن است از طریق یک یا چند مؤلفه دیگر انجام شود، مثلًاً به این دلیل که رابط مشتری و سرور همخوانی نداشته و بنابراین نیاز به نوعی تبدیل وجود دارد. دلیل دیگر می تواند این باشد که مؤلفه های متصل شده در ماشینهای مختلف قرار گرفته اند.

در Jade از ایده دامنه مدیریت تعمیر استفاده می شود. چنین دامنه ای متشکل از تعدادی گره است که هریک نماینده یک سرور و مؤلفه هایی است که توسط آن سرور اجرا می شود. یک مدیر گره جداگانه هم وجود دارد که مسئول افزودن گره ها و حذف آنها از دامنه است. ممکن است برای تضمین دسترسی بالا، مدیر گره کپی برداری شود.

هر یک از گره ها مجهز به کشف کننده خرابی هستند که بر سلامت گره یا یکی از مؤلفه های آن نظارت کرده و هر نوع خرابی را به مدیر گره گزارش می دهند. نوعاً، این کشف کننده ها تغییرات استثنایی در حالت مؤلفه، کاربرد منابع، و خرابی واقعی مؤلفه (این حالت ممکن است به این معنی باشد که یک ماشین از کارافتاده است) را مورد بررسی قرار می دهند.

پس از شناسایی خرابی روال تعمیر آغاز می شود. این روال بوسیله یک سیاست تعمیری گدانده می شود که بخشی از آن بوسیله مدیر گره اجرا می شود. سیاست ها به صورت صریح بیان شده و برحسب خرابی شناسایی شده اجرا می شوند. عنوان مثال، فرض کنید که خرابی در یک گره شناسایی شده و طبق سیاست تعمیر مراحل زیر باید انجام شود:

۱. توقف هر نوع پیوند بین مؤلفه ها روی یک گره غیرمعیوب و مؤلفه واقع برروی گرهی که به تازگی معیوب شده است.
۲. درخواست از مدیر گره جهت آغازه کار و افزودن گره جدید به دامنه.
۳. پیکربندی گره جدید با دقیقاً همان مؤلفه های گره از کار افتاده.
۴. برقراری مجدد تمامی پیوندهایی که قبلاً متوقف شده بودند.

<sup>288</sup> Interface

<sup>289</sup> Binding

در این مثال، سیاست تعمیر ساده بوده و فقط در صورتی قابل اجرا خواهد بود که داده های کلیدی گم نشده باشند) مؤلفه های از کارافتاده اصطلاحاً بی حالت<sup>۲۹۰</sup> نامیده می شوند).

روش مورد استفاده در Jade نمونه ای است از خود مدیریتی، یعنی اجرای اتوماتیک سیاست تعمیر به محض شناسایی خرابی به منظور بازگرداندن کل سیستم به وضعیت پیش از از کارافتادگی. در این سیستم که بر اساس مؤلفه است این نوع تعمیر اتوماتیک نیازمند نوعی پشتیبانی است تا قطعات بتوانند در زمان اجرا اضافه و حذف شوند. در مجموع باید گفت که تبدیل برنامه های کاربردی قدیمی به سیستم های خود مدیریتی غیرممکن است.

## ۲-۵ - خلاصه

سیستم های توزیعی را به روشهای مختلف می توان سازماندهی نمود. ما می توانیم بین معماری نرم افزار و معماری سیستم تمایز قائل شویم. در معماری سیستم محل قرارگرفتن مؤلفه های تشکیل دهنده سیستم توزیعی در کل ماشینهای مختلف درنظر گرفته می شود. در معماری نرم افزار بیشتر به بررسی سازمان منطقی نرم افزار می پردازد: یعنی نحوه تعامل مؤلفه های مختلف، شیوه های ساختاربندی آنها، چگونگی مستقل سازی آنها و غیره می پردازد.

یکی از مهم ترین ایده های مورد بحث در معماری ها بحث شیوه های معماری است. شیوه<sup>۲۹۱</sup> در واقع بیانگر اصول اساسی سازماندهی تعامل بین مؤلفه های نرم افزاری است که سیستم توزیعی را ساخته اند. از جمله مهمترین شیوه ها می توان به لایه بندی، شیء گرایی، رویداد گرایی و فضای-داده گرایی اشاره کرد.

سازمان های مختلفی برای سیستم های توزیعی وجود دارد. یک کلاس مهم آن است که ماشین ها به مشتری ها و سرورها تقسیم بندی می شوند. در این حالت، مشتری درخواستی را برای سرور ارسال می کند، سپس سرور نتیجه را تولید و آنرا به مشتری بازمی گرداند. معماری مشتری- سرور بیانگر روش مرسوم پیمانه ای سازی نرم افزار است که طی آن ، یک پیمانه مفروض اقدام به فراخوانی کارکرد های موجود در پیمانه دیگر می کند. با قراردادن مؤلفه های مختلف در ماشینهای مختلف، به نوعی توزیع فیزیکی و طبیعی از کارکردها در مجموعه ای از ماشینها دست خواهیم یافت.

معماری های مشتری- سرور غالباً بسیار مت مرکز هستند. در معماری های غیرمت مرکز، غالباً شاهد هستیم که فرآیندهای تشکیل دهنده سیستم توزیعی -که به نام سیستم های نظری به نظری هم خوانده می شوند- به یک اندازه نقش دارند. در سیستم های نظری به نظری ، فرآیندها به شکل یک شبکه فوقانی سازماندهی می شوند ؛ شبکه فوقانی در واقع یک شبکه منطقی است که هر یک از فرآیندهای آن، لیست محلی از فرآیندهای نظری دیگر را در اختیار دارد و می تواند با آنها ارتباط داشته باشد. شبکه فوقانی می تواند دارای ساختار باشد، به گونه ای که از طرح های مشخصی جهت مسیردهی پیام ها در بین

<sup>290</sup> Stateless

<sup>291</sup> Style

فرآیندها استفاده نماید. در شبکه های فاقد ساختار ، لیست فرایندهای نظیر کمابیش تصادفی است ؛ به این معنا که جهت مکان یابی داده ها یا دیگر فرآیندها نیازمند پیاده سازی الگوریتم های جستجو هستیم.

به عنوان جایگزین، سیستم های توزیعی خودمدیریتی پیاده سازی شده اند. در این سیستمها تا حدودی ایده هایی از معماری های نرم افزار و سیستم با هم ترکیب می شود. سیستم های توزیعی خودمدیریتی را می توان بصورت حلقه های کنترل بازخوردی سازمان دهی کرد. این قبیل حلقه ها، دارای یک مؤلفه نظارتی جهت اندازه گیری رفتار سیستم توزیعی ، یک مؤلفه آنالیزی جهت بررسی لزوم یا عدم لزوم تنظیمات و همچنین ، مجموعه ای از ابزارهای مختلف جهت تغییر رفتار می باشند. حلقه های کنترل بازخوردی را می توان در محل های مختلف در سیستم های توزیعی تلفیق نمود. تا رسیدن به درکی مشترک از نحوه ایجاد این حلقه ها و همچنین اجرای آنها به پژوهش بسیاری نیاز داریم.

### مسائل فصل

- ۱- چنانچه مشتری و سرور خیلی دور از هم قرار داده شوند، تأخیر شبکه تأثیر تعیین کننده ای بر کارآیی خواهد داشت. چه راه حلی برای این مشکل پیشنهاد می کنید؟
- ۲- معماری مشتری-سرور سه طبقه چیست.
- ۳- چه تفاوتی بین توزیع عمودی و توزیع افقی وجود دارد؟
- ۴- زنجیره ای از فرآیندها به صورت  $P_1, P_2, \dots, P_n$  را در نظر بگیرید که معماری چندطبقه ای مشتری- سرور را پیاده سازی می کنند. فرآیند  $P_i$  مشتری فرآیند  $P_{i+1}$  بوده و  $P_i$  فقط پس از دریافت پاسخ از  $P_{i+1}$ ، جواب را به  $P_{i-1}$  عودت خواهد داد. با بررسی عملکرد درخواست- پاسخ در فرآیند  $P_1$  مهمترین مشکلاتی را که در این سازمان می بینید ذکر کنید.
- ۵- در یک شبکه فوقانی دارای ساختار، پیام ها براساس توپولوژی شبکه فوقانی مسیریابی می شوند. یکی از مهمترین معایب این روش را ذکر کنید.
- ۶- شبکه CAN شکل ۲-۸ را درنظر بگیرید. برای مسیریابی پیام از گره با مختصات (۰،۰،۰) به گره دیگر با مختصات (۹،۰،۶) چگونه عمل می کنید؟
- ۷- با این فرض که گره واقع در CAN از مختصات همسایگان بلافصل خود اطلاع دارد، یکی از منطقی ترین سیاستهای مسیریابی می تواند ارسال پیام به نزدیکترین گره به سوی مقصد باشد. این سیاست چقدر خوب است.
- ۸- شبکه فوقانی بدون ساختاری را در نظر بگیرید که در آن هر گره بطور تصادفی تعداد  $C$  همسایه را انتخاب می کند. چنانچه  $P$  و  $Q$  هردو همسایه  $R$  باشند، احتمال این که همسایه یکدیگر باشند چقدر است؟

- ۹- مجدداً یک شبکه فوقانی بدون ساختار را در نظر بگیرید که در آن هر گره بطور تصادفی تعداد  $C$  همسایه را انتخاب می کند. برای جستجوی یک فایل، این گره سیل درخواست را به سوی همسایگان جاری کرده و از آنها می خواهد که درخواست را مجدداً به صورت سیل ارسال کنند. به چند گره خواهد رسید؟
- ۱۰- هرگرهی را نمی توان در سیستم نظیریه نظیر تبدیل به ابرنظریه نمود. یک گره برای این منظور چه ویژگی هایی باید داشته باشد؟
- ۱۱- یک سیستم BitTorrent را در نظر بگیرید که در آن هر گره دارای پیوند بیرونی با ظرفیت پهنهای باند  $B_{out}$  داشته باشد و هر پیوند ورودی دارای ظرفیت پهنهای باند  $B_{in}$  باشد. تعدادی از این گره ها (به نام بذر) داوطلبانه فایل هایی را برای دانلود کردن به دیگران ارائه می دهند. در صورتیکه فرض کنیم یک مشتری BitTorrent می تواند در هر زمان با حداکثر یک بذر تماس برقرار نماید، حداکثر ظرفیت دانلود مشتری BitTorrent چقدر خواهد بود؟
- ۱۲- دلیل متقاعد کننده (فنی) مطرح کنید که چرا سیاست تلافی جویانه (tit-for-tat) مورد استفاده در BitTorrent، در اشتراک فایل در اینترنت بهینه نمی باشد.
- ۱۳- دو مثال از کاربرد راه گیر ها در میان افزار تطبیقی ارائه شد. چند مثال دیگر از همین مورد ذکر کنید.
- ۱۴- راه گیرها تا چه اندازه وابسته به میان افزاری هستند که در آن مورد استفاده قرار می گیرند؟
- ۱۵- در خودروهای مدرن با انواع ابزارهای الکترونیک مواجه هستیم. نمونه هایی از سیستم های کنترل بازخوردی را در این خودروها ذکر کنید.
- ۱۶- نمونه ای از یک سیستم خود مدیریتی ذکر کنید که در آن مؤلفه آنالیز کننده بطور کامل توزیع یا حتی مخفی شده است.
- ۱۷- راه حلی برای تعیین اتوماتیک بهترین طول مسیر (Trace) برای پیش بینی سیاستهای کپی برداری در Globule ذکر کنید.
- ۱۸- (تکلیف آزمایشگاهی) بكمک نرم افزار موجود، یک سیستم بربایه BitTorrent جهت توزیع فایل ها در مشتریان متعدد از یک سرور واحد و قدرتمند طراحی و پیاده سازی کنید. برای سهولت کار می توانید از یک سرور وب استاندارد که می تواند به عنوان ردیاب (Tracker) عمل کند استفاده کنید.

### ۳ ۲۹۲ فرآیندها

در این فصل، نگاه دقیقتری به نقش کلیدی فرآیندهای مختلف در سیستم‌های توزیعی خواهیم داشت. مفهوم فرآیند ریشه در حوزه سیستم عامل داشته و غالباً بصورت برنامه در حال اجرا تعریف می‌شود. از نقطه نظر سیستم عامل، مدیریت و زمان بندی فرآیندها را می‌توان از مهم ترین مسائل مورد بحث به حساب آورد. در حالیکه در سیستم‌های توزیعی، مسائل دیگر از اهمیت برابر یا بیشتری برخوردار هستند.

بعنوان مثال، جهت سازماندهی مؤثر سیستم‌های مشتری-سرور، غالباً بهتر است که از تکنیک‌های چندنخی استفاده کنیم. همانطور که در بخش اول همین فصل هم خواهیم گفت، یکی از مهم ترین نقش‌های نخ<sup>۲۹۳</sup>‌ها در سیستم‌های توزیعی آن است که باعث می‌شوند تا مشتریان و سرورها به صورتی ایجاد شوند که ارتباطات و پردازش محلی با هم هم پوشانی کرده و همین امر باعث افزایش سطح کارآیی آنها خواهد شد.

در سالهای اخیر، مفهوم مجازی سازی<sup>۲۹۴</sup> کاربرد گسترده‌ای پیدا کرده است. مجازی سازی به برنامه کاربردی، وشاید به محیط کامل آن و از جمله خود سیستم عامل، امکان می‌دهد تا بصورت همزمان با سایر برنامه‌های کاربردی اجرا شود. اما اینکار بصورتی کاملاً مستقل از سخت افزار و سکو<sup>۲۹۵</sup>‌های زیرین انجام شده و همین امر منجر به درجه بالایی از جابجاپذیری<sup>۲۹۶</sup> در آنها خواهد شد. بعلاوه، مجازی سازی به تفکیک و مجزاسازی خرابی‌های ناشی از خطاهای و مشکلات امنیتی کمک می‌کند. مجازی سازی یکی از مفاهیم مهم در سیستم‌های توزیعی بوده و به همین دلیل بخش مجازی را به آن اختصاص داده ایم.

همانطور که در فصل ۲ هم گفتیم، سازمان‌های مشتری-سرور در سیستم‌های توزیعی از اهمیت بالایی برخوردار هستند. در این فصل، نگاه دقیق‌تری خواهیم داشت به برخی سازمان بندی‌های نمونه مشتری و سرور. همچنین راجع به مسائل عمومی طراحی سرورها صحبت خواهیم کرد.

یکی از مهمترین مسائل، خصوصاً در سیستم‌های توزیعی حوزه گسترده<sup>۲۹۷</sup>، جابجایی فرآیندها در بین ماشین‌های مختلف است. مهاجرت فرآیند<sup>۲۹۸</sup> یا به بیان مشخص‌تر، مهاجرت کدها<sup>۲۹۹</sup>، می‌تواند هم در

<sup>292</sup> processes

<sup>293</sup> thread

<sup>294</sup> Virtualization

<sup>295</sup> Platform

<sup>296</sup> portability

<sup>297</sup> wide-area

<sup>298</sup> process migration

<sup>299</sup> code migration

کسب قابلیت مقیاس پذیری<sup>۳۰۰</sup> مفید واقع شود و هم در پیکربندی پویای مشتری‌ها و سرورها، مفهوم دقیق مهاجرت کد و اثرات جانبی آن هم در این فصل مورد بررسی قرار گرفته‌اند.

### ۱-۳- نخ‌ها

هرچند فرآیندها یکی از ستونهای تشکیل هنده سیستم‌های توزیعی محسوب می‌شوند، در عمل مشاهده شده که دانه بندی<sup>۳۰۱</sup> فرآیندهای ایجاد شده بوسیله سیستم‌های عاملی که مبنای سیستم‌های توزیعی را تشکیل می‌دهند، هنوزکافی نیست. پس به این نتیجه می‌رسیم که کسب دانه بندی طریف‌تر از طریق افزایش تعداد نخ‌ها در هر فرآیند ساخت برنامه‌های کاربردی توزیعی را آسانتر و باعث افزایش کارآیی می‌شود. در این بخش، نگاه دقیق‌تری به نقش نخ‌ها در سیستم‌های توزیعی داشته و دلیل اهمیت آنها را توضیح خواهیم داد. اطلاعات بیشتر در مورد نخ‌ها و نحوه استفاده از آنها برای ایجاد برنامه‌های کاربردی را در Stevens(1999) و Lewis and Berg(1998) مطالعه کنید.

### ۱-۳- مقدمه‌ای بر نخ‌ها

برای درک نقش و اهمیت نخ‌ها در سیستم‌های توزیعی، باید بدانیم که فرآیند چیست و فرآیندها و بندها چگونه با هم ارتباط پیدا می‌کنند. سیستم عامل برای اجرای برنامه تعدادی پردازنده مجازی ایجاد کرده و هریک را مسئول اجرای برنامه جداگانه‌ای می‌کند. برای ردگیری این پردازنده‌های مجازی سیستم عامل یک جدول فرآیند<sup>۳۰۲</sup> در اختیار دارد؛ این جدول حاوی ورودی‌هایی جهت ذخیره مقادیر ثبات‌های پردازنده، نگاشت‌های حافظه، فایل‌های باز، اطلاعات حسابرسی، الیت‌ها و غیره می‌باشد. فرآیند هم غالباً به صورت برنامه در حال اجرا - یعنی برنامه‌ای که هم اکنون روی یکی از همین پردازنده‌های مجازی سیستم عامل در حال اجراست - تعریف می‌شود. یکی از مسائل مهم این است که سیستم عامل بسیار مراقب است تا مبادا فرآیندهای مستقل سهواً یا عمدتاً کوچکترین تأثیری بر صحت رفتار یکدیگر بگذارد. به بیان دیگر، این واقعیت که چندین برنامه می‌توانند به صورت همرونند در یک پردازنده واحد یا دیگر منابع سخت افزاری شریک شوند، شفاف سازی شده است.

سیستم عامل برای موقوفیت در تفکیک فرآیندها، معمولاً نیازمند پشتیبانی سخت افزار است.

برای دستیابی به این شفافیت همرونندی باید هزینه نسبتاً گزافی پرداخت کرد. بعنوان مثال، هرزمان که فرآیندی ایجاد می‌شود، سیستم عامل باید یک فضای آدرس کالاً مستقل به ازای آن ایجاد کند. در اینجا تخصیص می‌تواند به معنای مقدار دهی اولیه حافظه باشد، مثلًاً صفر قرار دادن یک قطعه داده ای، کپی کردن برنامه مربوطه به قطعه متنی، و ایجاد پشته برای داده‌های موقت. به همین ترتیب، سوئیچ کردن پردازنده بین دو فرآیند هم ممکن است تا حدودی هزینه بر باشد. جدای از ذخیره متن

<sup>300</sup> scalability

<sup>301</sup> Granularity

<sup>302</sup> process table

پردازنده (شامل مقادیر ثبات ها، برنامه شمار، اشاره گر پشته، وغیره) سیستم عامل مجبور به تغییر ثبات های واحد مدیریت حافظه<sup>۳۰۳</sup> (MMU) و نامعتبر سازی حافظه های پنهان ترجمه آدرس مانند بافر دم دستی ترجمه<sup>۳۰۴</sup> (TLB) نیز می باشد. بعلاوه ، چنانچه تعداد فرآیندهای تحت پشتیبانی سیستم عامل بیش از آن چیزی باشد که می تواند بصورت همزمان در حافظه اصلی نگه دارد، ممکن است مجبور باشد تا پیش از سوئیچ بین فرآیندها، فرآیندهایی را بین حافظه اصلی و دیسک مبادله کند. مشابه فرآیند، نخ هم قطعه کد مخصوص به خود را، به صورتی مستقل از دیگر نخ ها، اجرا می کند. اما برخلاف فرآیندها، چنانچه کسب درجه بالایی از شفافیت همروندی منجر به افت کارآیی شود، هیچ تلاشی برای دستتابی به آن انجام نمی شود . بنابراین، یک سیستم نخ معمولاً فقط حداقل اطلاعات لازم برای ایجاد امکان اشتراک پردازنده در بین چندین نخ را در خود نگه می دارد. علی الخصوص اینکه، متن نخ<sup>۳۰۵</sup> غالباً فقط شامل متن پردازنده به همراه مختصر اطلاعات دیگری جهت مدیریت نخ است. بعنوان مثال، یک سیستم نخ ممکن است فقط پی گیری کند که در حال حاضر روی یک متغیر انحصار متقابل<sup>۳۰۶</sup> مسدود شده است تا به این ترتیب مانع از انتخاب آن برای اجرا شود. آن دسته از اطلاعاتی که ضرورت چندانی در مدیریت چندنخی نداشته باشند، غالباً نادیده گرفته می شوند. به همین دلیل، محافظت از داده ها در برابر دسترسی نامناسب بوسیله نخ ها در هر فرآیند بطور کامل بر عهده پدید آورندگان برنامه های کاربردی قرار دارد.

این روش دو اثر جانبی مهم دارد. در درجه اول، کارآیی یک برنامه کاربردی چندنخی چندان ضعیفتر از عملکرد رقیب تک نخی خود نخواهد بود . در واقع، در بسیاری از موارد چند نخی شدن منجر به افزایش کارآیی هم می شود. ثانیاً، از آنجایی که نخ ها به صورت اتوماتیک در برابر یکدیگر به آن صورتی که فرآیندها محافظت می شوند محافظت نشده اند، ایجاد برنامه های کاربردی چند نخی نیاز به تلاش فکری بیشتری دارد. در اینجا هم ، طراحی مناسب و ساده سازی امور بسیار مفید خواهد بود. متأسفانه ، از اوضاع فعلی می توان نتیجه گرفت که این اصل هنوز به خوبی درک و مورد استفاده قرار نگرفته است.

### کاربرد نخ در سیستم های غیرتوزیعی

پیش از بحث در مورد نقش نخ ها در سیستم های توزیعی ، اجازه دهید ابتدا در مورد کاربرد آنها در سیستم های متعارف غیر توزیعی صحبت کنیم. فرآیندهای چند نخی مزایای متعددی دارند و همین ویژگی باعث افزایش تمایل نسبت به کاربرد آنها شده است.

مهم ترین مزیت ناشی از این واقعیت است که در یک فرآیند تک نخی ، با اجرای فراخوان سیستمی مسدود شونده، کل فرآیند مسدود خواهد شد. برای درک بهتر موضوع، یک برنامه کاربردی مانند یک

<sup>303</sup> Memory Management Unit

<sup>304</sup> Translation Lookaside Buffer

<sup>305</sup> thread context

<sup>306</sup> mutex

برنامه صفحه گسترده را تصور کنید که کاربر بخواهد بطور دائم و به صورت تعاملی مقادیر را تغییر دهد. یکی از ویژگی های مهم برنامه های صفحه گسترده این است که وابستگی های کارکردی بین سلول های مختلف را حتی از صفحه گسترده های دیگر هم نگه می دارد. بنابراین، با تغییر یک سلول ، تمامی سلولهای وابسته هم بصورت اتوماتیک به هنگام خواهند شد. وقتی کاربر مقدار عددی داخل یک سلول را تغییر می دهد، این اصلاح می تواند باعث ایجاد چندین سری محاسبات شود. در صورت وجود فقط یک نخ کنترل، کار محاسبه نمی تواند همزمان با انتظار برنامه برای ورودی انجام شود. به همین ترتیب، نمی توان در حین محاسبه وابستگی ها، اقدام به ایجاد ورودی نمود. ساده ترین راه حل این است که حداقل از دو نخ کنترلی استفاده کنیم، یکی برای هدایت جریان تعاملی با کاربر و دیگری برای به هنگام سازی صفحه گسترده. در این حین می توان ، همزمان با فعال و در حال کار بودن دو نخ، از نخ سومی هم برای پشتیبانی از صفحه گسترده روی دیسک استفاده نمود.

یکی دیگر از مزایای چند نخی این است که امکان استفاده از موازی گرایی<sup>۳۰۷</sup> را به هنگام اجرای برنامه روی یک سیستم چند پردازنده ای فراهم می آورد . در این حالت، هر نخ به یک پردازنده متفاوت نسبت داده شده و داده های اشتراکی هم در حافظه اصلی اشتراکی ذخیره می شوند. این موازی گرایی در صورتیکه با طراحی مناسب همراه شود، می تواند شفاف باشد؛ یعنی فرآیند می تواند به همان راحتی اما با سرعتی کمتر، روی یک سیستم تک پردازنده اجرا شود. به دلیل افزایش میزان دسترسی به ایستگاههای کاری چندپردازنده ای نسبتاً ارزان قیمت ، چند نخی برای موازی گرایی هم اهمیتی بیش از پیش یافته است. از این دست سیستم های کامپیوتری نوعاً جهت سورها در برنامه های کاربردی مشتری-سرور استفاده می شود.

از مزایای دیگر چند نخی می توان به کاربرد مفید آنها در متن برنامه های کاربردی بزرگ اشاره کرد. این نوع برنامه های کاربردی غالباً بصورت مجموعه ای از برنامه های همکاری کننده ایجاد می شوند که هریک بایستی توسط یک فرآیند جداگانه اجرا شوند. این روش خصوصاً در محیط UNIX کاربرد دارد. همکاری بین برنامه ها بوسیله مکانیزم های ارتباط میان فرآیندی<sup>۳۰۸</sup> (IPC) اجرا می شود. در سیستم UNIX، این مکانیزم ها غالباً شامل لوله ها<sup>۳۰۹</sup> (نامگذاری شده)، صفحه های پیام ها، و قطعه های حافظه اشتراکی می باشد] همچنین مراجعه شود به (۲۰۰۵) [Stevens and Rago]. مهمترین نقطه ضعف تمامی مکانیزم های IPC آن است که ارتباط غالباً مستلزم سوئیچ کردن متن به مقدار زیاد است که در قالب سه نقطه مختلف در شکل ۱-۳ نمایش داده شده است.

<sup>307</sup> parallelism

<sup>308</sup> Interprocess Communication

<sup>309</sup> Pipes

شکل ۱-۳- سوئیچ کردن متن در نتیجه IPC (داخل شکل: فرآیند A، فرآیند B) سوئیچ کردن از فضای کاربر به فضای هسته اصلی، S2: سوئیچ کردن بافت از فرآیند A به فرآیند B، S3: سوئیچ کردن از فضای هسته اصلی به فضای کاربر).

از آن جایی که IPC نیازمند دخالت هسته اصلی است، غالباً لازم است که فرآیند ابتدا از حالت کاربر به حالت هسته اصلی سوئیچ کند که در شکل ۳-۱ با S1 نمایش داده شده است. این کار هم مستلزم تغییر نگاشت حافظه<sup>۳۱۰</sup> در واحد مدیریت حافظه (MMU) و همچنین تخلیه TLB است. در داخل هسته اصلی، سوئیچ متن فرآیند انجام می شود (S2 در شکل فوق) و سپس طرف دیگر با تغییر از حالت هسته اصلی به حالت کاربر فعال می شود (S3 شکل ۱-۳). تغییر اخیر هم مستلزم تغییر نگاشت MMU و تخلیه TLB است.

به جای استفاده از فرآیندها، می توان برنامه کاربردی ایجاد نمود که بخش های مختلف آن بوسیله نخ های مجزا اجرا شوند. ارتباط بین این بخش ها تماماً با استفاده از داده های مشترک صورت می پذیرد. سوئیچ بین نخ ها ممکن است بعضاً بطور کامل در فضای کاربر انجام شود. با این وجود در بسیاری از پیاده سازی ها، هسته اصلی از نخ ها آگاه بوده و آنها را زمان بندی می کند. این امر ممکن است منجر به ارتقاء چشمگیر کارآیی شود.

دلیل آخر برای استفاده از نخ ها کاملاً مربوط به مهندسی نرم افزار است: ساخت بسیاری از برنامه های کاربردی صرفاً بصورت مجموعه ای از نخ های همکاری کننده بسیار آسانتر است. بعنوان نمونه برنامه های کاربردی را در نظر بگیرید که باید امور (کمابیش مستقل) متعددی را انجام دهند. مثلاً در مورد پردازشگر های کلمه<sup>۳۱۱</sup>، می توان از نخ های جداگانه برای کاربروی ورودی کاربر، چک کردن گرامر و املاء، صفحه بندی سند، تهیه فهرست موضوعی و غیره استفاده کرد.

### پیاده سازی نخ

نخ ها غالباً به شکل یک بسته ارائه می شوند. این بسته شامل عملیاتی جهت ایجاد و از بین بردن نخ ها و همچنین عملیات روی متغیرهای همگام سازی<sup>۳۱۲</sup> از قبیل متغیر انحصار متقابل (mutex) و متغیرهای شرایطی است. اساساً دو روش برای پیاده سازی بسته نخ وجود دارد. روش اول ساخت کتابخانه نخ است که کاملاً در حالت کاربر اجرا شود. در روش دوم، هسته اصلی آگاه از وجود نخ ها و زمان بندی آنها است.

کتابخانه نخ سطح کاربر چند مزیت دارد. اول، ایجاد و از بین بردن نخ ها ارزان است. از آنجا که تمام مدیریت نخ در فضای آدرس کاربر نگهداری می شود، هزینه ایجاد نخ اساساً همان هزینه تخصیص حافظه برای تهیه یک پشته برای نخ است. به همین ترتیب، از بین بردن نخ عمدهاً شامل آزاد سازی

<sup>310</sup> Memory map

<sup>311</sup> Word Processor

<sup>312</sup> Synchronization

حافظه تخصیص داده شده برای پشته نخ است که دیگر استفاده ندارد. هردو عملیات فوق کم هزینه هستند.

دومین مزیت نخ های سطح کاربر آن است که سوئیچ کردن متن نخ فقط بوسیله چند دستورالمعل قابل انجام است . اساساً، فقط باید مقادیر ثبات های پردازنده را ذخیره کرد و سپس مقادیر قبلًا ذخیره شده نخی که به آن سوئیچ می شود مجدداً بارگذاری کرد. هیچ لزومی به تغییر نگاشت های حافظه، تخلیه TLB، انجام حسابرسی پردازنده و از این قبیل وجود ندارد. تغییر متن نخ در صورت لزوم همگام شدن دو نخ ، مثلًا زمان ورود به بخشی از داده های مشترک مطرح می شود.

با این وجود، نقطه ضعف عمدۀ نخ های سطح کاربر آن است که فراخوان سیستمی مسدود شونده فوراً باعث مسدود شدن کل فرآیندی خواهد شد که این نخ و تمامی نخ های دیگر موجود در فرآیند به آن تعلق دارند . همان طور که قبلًا هم توضیح دادیم، اهمیت ویژه نخ ها در تبدیل برنامه های کاربردی بزرگ به صورت قطعات منطقی قابل اجرای هم زمان است. در این حالت، مسدود شدن روی ورودی-خروجی قاعدها نباید مانع از اجرای قطعات دیگر در این زمان باشد. در چنین برنامه های کاربردی، استفاده از نخ های سطح کاربر مفید نخواهد بود.

اغلب این مشکلات را می توان با پیاده سازی نخ ها در هسته اصلی سیستم عامل حل و فصل نمود. متأسفانه، هزینه ای که باید بپردازیم بالاست: تمامی عملیات نخ ( از قبیل ایجاد، حذف، همگام سازی و غیره) بایستی بوسیله هسته اصلی اجرا شده و این امر مستلزم فراخوانی سیستمی است. در اینصورت، تغییر متن نخ ممکن است به اندازه تغییر متن فرآیند هزینه بر باشد. در نتیجه، اغلب مزایای عملکردی استفاده از نخ ها بجای فرآیندها از بین خواهد رفت.

یک راه حل استفاده از صورت ترکیبی نخ های سطح کاربر و سطح هسته اصلی است که غالباً به نام **فرآیندهای سبک وزن<sup>۳۱۳</sup> (LWP)** خوانده می شود. LWP در متن یک فرآیند (سنگین وزن) اجرا شده و ممکن است در هر فرآیند چندین LWP وجود داشته باشد. هر سیستم علاوه بر دارا بودن LWP ها ، یک بسته نخ سطح کاربر را هم ارائه می دهد- این بسته به برنامه های کاربردی، عملیات معمول برای ایجاد و از بین بردن نخ ها ارائه می دهد. علاوه ، این بسته امکاناتی از قبیل متغیر انحصار متقابل (mutex) و متغیرهای شرطی را برای همگام سازی نخ ها ارائه می دهد. مسئله مهم این است که بسته نخ تمامًا در فضای کاربر اجرا می شود. به بیان دیگر، تمامی عملیات روی نخ ها بدون دخالت هسته اصلی انجام می شود.

شکل ۲-۳- ترکیب فرآیندهای سبک وزن سطح هسته اصلی و نخ های سطح کاربر( فضای کاربر، وضعیت نخ، نخ، فضای هسته اصلی، LWP در حال اجرای یک نخ ، فرآیند سبک وزن).

<sup>313</sup> Lightweight Processes

مطابق شکل ۳-۲ ، یک بسته نخ ممکن است در بین چندین فرآیند LWP مشترک باشد . به این معنا که هر LWP می تواند نخ ( سطح کاربر) ویژه خود را اجرا کند. برنامه های کاربردی چند نخی از طریق ایجاد نخ ها و سپس، تخصیص هریک از نخ ها به یک LWP ساخته می شود. تخصیص نخ به LWP عمدتاً ضمنی و از برنامه نویس مخفی می باشد.

در ترکیب نخ ها(ی سطح کاربر) و LWP ها به این ترتیب عمل می شود: بسته نخ برای زمان بندی نخ بعدی فقط یک روال در اختیار دارد. در هنگام ایجاد LWP (که با فراخوان سیستمی انجام می شود)، پشته خاص LWP در اختیار آن قرار گرفته و به او دستور داده می شود که برای جستجوی نخ مورد نظر برای اجرا، روال زمان بندی را اجرا کند. در صورت تعدد LWP ها، هریک از آنها زمان بند را اجرا می کند. بنابراین جدول نخ ها، که برای نگهداری گزارش مجموعه نخ های فعلی مورد استفاده قرار می گیرد، در بین LWP ها به اشتراک گذاشته می شود. محافظت از این جدول جهت تضمین دسترسی به صورت انحصاری بوسیله متغیر های انحصار متقابلی انجام می شود که تماماً در فضای کاربر بیان سازی می شوند. به بیان دیگر، همگام سازی در بین LWP ها مستلزم هیچ نوع پشتیبانی از جانب هسته اصلی نمی باشد.

پس از آنکه LWP نخ قابل اجرایی را پیدا کرد، متن را به آن نخ سوئیچ می کند. ممکن است در این فاصله زمانی، LWP های دیگری هم در جستجوی نخ های قابل اجرای دیگر باشند. چنانچه لازم باشد نخی روی یک متغیر انحصار متقابل یا روی یک متغیر شرطی مسدود شود، مدیریت اجرایی لازمه را انجام داده و نهایتاً روال زمان بندی را فراخوانی می کند. با پیدا شدن نخ قابل اجرای دیگر، تغییر متن به آن انجام خواهد شد. زیبایی همه اینها در این است که نیازی به مطلع ساختن LWP اجرا کننده نخ وجود ندارد ؛ یعنی تغییر متن تماماً در فضای کاربر انجام شده و برای LWP بصورت یک کد برنامه عادی به نظر می رسد.

حال بینیم با فراخوانی سیستمی مسدود شونده توسط نخ چه اتفاقاتی رخ می دهد. در اینجا حالت اجرا از حالت کاربر به حالت هسته اصلی تغییر می یابد، اما بازهم در متن LWP فعلی ادامه می یابد. در نقطه ای که LWP فعلی دیگر قادر به ادامه کار نباشد، ممکن است سیستم عامل تصمیم به تغییر متن به LWP دیگری بگیرد. این امر ضمناً به معنای بازگشت حالت اجرایی به حالت کاربر است. LWP انتخاب شده درست از جایی که قبلًا قطع شده بود، ادامه خواهد یافت.

استفاده از صورت ترکیبی LWP ها و بسته نخ سطح کاربر مزایای متعددی دارد. اولاً، ایجاد، از بین بردن و همگام سازی نخ ها نسبتاً کم هزینه بوده و به هیچ وجه نیازمند دخالت هسته اصلی نمی باشد. ثانیاً، مشروط بر این که فرآیندی LWP های کافی در اختیار داشته باشد، فراخوانی سیستمی مسدود شونده باعث تعليق کل فرآیند نخواهد شد. ثالثاً، نیازی به اطلاع برنامه کاربردی از LWP ها وجود ندارد، بلکه این برنامه فقط با نخ های سطح کاربر سروکار دارد. رابعاً، می توان با اجرای LWP های مختلف بر روی پردازنده های مختلف، به راحتی از آنها در محیط های چندپردازنده ای استفاده نمود. این

چندپردازشی می تواند کاملاً از برنامه کاربردی مخفی باشد. تنها نقطه ضعف ترکیب فرآیندهای سبک وزن با نخ های سطح کاربر این است که همچنان نیازمند ایجاد و از بین بردن LWP ها هستیم که مانند نخ های سطح هسته اصلی هزینه زیادی دارد. با این وجود، ایجاد و از بین بردن LWP ها فقط به ندرت انجام شده و غالباً بطور کامل تحت کنترل سیستم عامل قرار دارد.

بعنوان روش دیگر، که مشابه رویکرد فرآیندهای سبک وزن می باشد، می توان به استفاده از **فعال سازی زمان بند**<sup>۳۱۴</sup> اشاره کرد (Anderson، ۱۹۹۱) و گروه همکاران). عمدۀ ترین تفاوت بین **فعال سازی زمان بند** و LWP آن است که وقتی یک نخ در اثر فراخوانی سیستم مسدود می شود، هسته اصلی روال زمان بند را جهت انتخاب نخ قابل اجرای بعدی فراخوانی می کند. همین روال در صورت خارج شدن از انسداد تکرار خواهد شد. مزیت روش مذکور آن است که مدیریت LWP بوسیله هسته اصلی انجام نمی شود. با این وجود، استفاده از روش فراخوانی به سمت بالا<sup>۳۱۵</sup> خیلی جالب نیست، زیرا باعث برهم خوردن ساختار سیستم های لایه بندي می شود که در این سیستم ها فقط فراخوانی به لایه سطح پایین تر بعدی مجاز است.

### ۲-۳- نخ ها و سیستم های توزیعی

یکی از مهمترین ویژگی های نخ ها آن است که می توانند بدون مسدود کردن کل فرآیندی که نخ در آن در حال اجراست، راهکار مناسبی جهت فراخوانی های سیستمی مسدود شونده مهیا کنند. این ویژگی باعث جذابیت ویژه کاربرد نخ ها در سیستم های توزیعی می شود، چون به این ترتیب بیان ارتباط به صورتی همزمان با حفظ اتصالات منطقی متعدد بسیار آسانتر خواهد شد. در ادامه سعی خواهیم کرد تا با بررسی دقیق‌تر مشتری ها و سرورهای چندنخی این موضوع را تشریح کنیم.

#### مشتری های چندنخی

ممکن است لازم باشد تا برای دستیابی به شفافیت توزیعی بیشتر، سیستم های توزیعی که در شبکه های حوزه گسترده عمل می کنند، زمان های انتشار پیام های میان فرآیندی طولانی را مخفی کنند. تأخیر سفر گردشی<sup>۳۱۶</sup> در شبکه حوزه گسترده ممکن است به اندازه صدها میلی ثانیه و حتی صدها ثانیه باشد.

روش معمول در مخفی سازی تأخیرهای ارتباطی، آغاز ارتباط و ادامه بی درنگ کار دیگر است. یکی از کاربردهای این حالت در جستجوگرهای وب است. در بسیاری از موارد، سند وب متشکل از یک فایل HTML است که شامل یک متن عادی به همراه مجموعه ای از تصاویر، آیکن ها و غیره می باشد. برای واکشی هریک از مؤلفه های سند وب، جستجوگر باید با ایجاد یک اتصال TCP/IP داده های ورودی را بخواند و آنرا به مؤلفه صفحه نمایش عبور دهد. ایجاد اتصال و خواندن داده های ورودی ذاتاً

<sup>314</sup> Scheduler Activation

<sup>315</sup> Upcall

<sup>316</sup> Round-trip delay

عملیات مسدود شونده هستند. در هنگام بحث در مورد ارتباط در مسافت طولانی هم با این اشکال مواجه هستیم که ممکن است زمان تکمیل هر یک از عملیات نسبتاً طولانی باشد.

جستجوگرهای وب غالباً کار را با واکشی صفحه HTML آغاز و سپس آنرا نمایش می دهند. جهت مخفی سازی حداکثری تأخیرهای ارتباطی، برخی جستجوگرها کار نمایش داده ها را همزمان با ورود آنها آغاز می کنند. وقتی متن به همراه امکانات مرور کردن و دیگر امکانات در اختیار کاربر قرار گرفت، جستجوگر کار را با واکشی دیگر فایل های سازنده صفحه از قبیل تصاویر ادامه خواهد داد. تصاویر بصورت همزمان با ورود نمایش داده می شوند. بنابراین دیگر لازم نیست که کاربر منتظر بماند تا تمامی مؤلفه های کل صفحه واکشی شده و سپس صفحه در اختیار او قرار بگیرد.

در عمل شاهد بوده ایم که جستجوگرهای وب چند کار را به صورت همزمان انجام می دهند. بنابراین، ایجاد جستجوگر بعنوان یک مشتری چندنحوی شده تاحدود زیادی باعث تسهیل مسائل خواهد شد. به محض واکشی فایل HTML اصلی، می توان جهت واکشی سایر بخش ها، نخ های دیگری را هم فعال نمود. هر یک از نخ ها اتصال جداگانه ای را با سرور برقرار کرده و داده ها را به داخل آن می آورد. ایجاد اتصال و خواندن داده ها از سرور را می توان بكمک فراخوانی های سیستمی استاندارد (مسدود شونده) و با این فرض که فراخوانی مسدود شده باعث تعليق کل فرآيند خواهد شد، برنامه ریزی نمود. همان طور که در مرجع (Stevens ۱۹۹۸) هم تشریح شده، تمامی نخ ها از کد واحد و، مهمتر از همه این که، ساده ای برخوردار هستند. در همین حال، کاربر فقط متوجه تأخیر در نمایش تصاویر و نظایر آن شده و می تواند اقدام به جستجو در کل سند نماید.

مزیت مهم دیگری در استفاده از جستجوگرهای چند نخی وب آن است که می توان چندین اتصال را به صورت همزمان باز نمود. در مثال قبل، چندین اتصال به یک سرور واحد انجام شده بود. چنانچه این سرور دارای بار زیاد و یا بسیار کند باشد، روش چند نخی هیچ بهبود واقعی در کارآیی نسبت به دستیابی ترتیبی به فایل ها خواهد داشت.

با این وجود، در بسیاری از موارد، سرورهای وب در بین چندین ماشین تکرار شده و سرورهای هریک از آنها مجموعه دقیقاً واحدی از اسناد وب را مهیا می کند. سرورهای تکرار شده در سایت واحدی قرار داده شده و با نام واحدی هم خوانده می شوند. وقتی درخواستی در رابطه با یک صفحه وب وارد می شود، این درخواست غالباً براساس راهکار نوبت گردشی<sup>۳۱۷</sup> یا روش متعادل کننده بار دیگری (۱۹۹۴) Katz و گروه همکاران) به یکی از سرورها فرستاده می شود. در صورت استفاده از مشتری چند نخی ممکن است اتصالاتی به کپی های مختلف ایجاد شود. در نتیجه داده ها بصورت موازی انتقال پیدا کرده و همین امر به نحو چشمگیری باعث کاهش کلی در زمان نمایش سند وب در قیاس با سرور تکرار نشده خواهد شد. استفاده از این شیوه فقط در صورتی امکانپذیر خواهد بود که مشتری بتواند با

<sup>۳۱۷</sup> round-robin

جريانات واقعاً موازی داده های ورودی کار کند. برای این منظور نخ ها بهترین و مناسب ترین گزینه خواهند بود.

### سورهای چندنخی

هر چند مشتری های چند نخی مزایای قابل ملاحظه ای دارند، اما همانطور که مشاهده کردیم، کاربرد عمده چندنخی شدن در سیستم های توزیعی در طرف سرور می باشد. در عمل مشاهده شده که چندنخی شدن علاوه بر ساده سازی چشمگیر کد سرور، باعث سهولت فراوان ایجاد سرورهایی می شود که از ویژگی موازی گرایی برای افزایش کارآبی خود، حتی در سیستم های تک پردازنده ای، استفاده می کنند. با این وجود، حال که کامپیوترهای چندپردازنده ای در نقش ایستگاههای کاری چند منظوره کاربرد گسترده ای پیدا کرده اند، چندنخی شدن جهت موازی گرایی بیش از پیش کاربرد و پیدا کرده است.

برای درک مزایای استفاده از نخ ها در نوشتن کد سرور، سازمان سرور فایلی را در نظر بگیرید که برحسب مورد باید برای انتظار بر روی دیسک مسدود شود. سرور فایل معمولاً منتظر ورود درخواست برای عملیات فایل باقی می ماند. پس از ورود درخواست آن را اجرا کرده و در پایان پاسخ را باز می گرداند. یک سازمان ممکن و بسیار متداول در شکل ۳-۳ نمایش داده شده است. در اینجا یک نخ به نام پخش کننده<sup>۳۱۸</sup> اقدام به خواندن درخواست های ورودی برای عملیات فایل می کند. درخواست ها بوسیله مشتریان به نقطه انتهایی<sup>۳۱۹</sup> شناخته شده ای برای سرور مذکور ارسال می شوند. پس از بررسی درخواست، سرور یک نخ کارگر<sup>۳۲۰</sup> بی کار را انتخاب و درخواست را به آن تقدیم می کند.

شکل ۳-۳-یک سرور چند نخی که به شکل مدل پخش-کننده/ کارگر سازمان داده شده است (نخ پخش کننده، درخواست داده شده به نخ کارگر، سرور، نخ کارگر، درخواست ورودی از شبکه، سیستم عامل).

کارگر با انجام خواندن مسدود شونده روی سیستم فایل محلی ادامه می دهد، و اینکار ممکن است باعث تعليق نخ تا زمان واکشی داده ها از دیسک شود. در صورت معلق بودن نخ، نخ دیگری برای اجرا انتخاب خواهد شد. بعنوان مثال، ممکن است پخش کننده برای انجام کار بیشتر انتخاب شود. البته ممکن است نخ کارگر دیگری که هم اکنون آماده اجرا باشد انتخاب شود.

حال ببینیم سرور فایل در غیاب نخ ها چگونه نوشته می شود. یک راه حل این است که سرور بعنوان تک نخ عمل کند. حلقه اصلی سرور فایل درخواست را گرفته، آنرا بررسی می کند و پیش از گرفتن درخواست بعدی اقدام به تکمیل آن می کند. سرور در حین انتظار برای دیسک، بیکار بوده و هیچ درخواست دیگری را پردازش نمی کند. در نتیجه، نمی توان روی درخواستهای مشتریان دیگر کار کرد.

<sup>318</sup> dispatcher

<sup>319</sup> end point

<sup>320</sup> worker thread

بعلاوه، چنانچه سرور فایل روی ماشین اختصاص یافته در حال اجرا باشد - که معمولاً هم همینطور است - در فاصله زمانی که سرور فایل منتظر دیسک است، پردازنده کاملاً بیکار خواهد بود. درنتیجه ، تعداد درخواست های پردازش شده در ثانیه افت خواهد کرد. بنابراین نخ ها کارآیی بالایی دارند، ولی هرنخ به صورت ترتیبی و به طور معمول برنامه ریزی خواهد شد.

بنابراین تا به اینجا با دو نوع طراحی مواجه شده ایم : سرور فایل چندنخی و سرور فایل تک نخی. اما حالتی را فرض کنید که چند نخی موجود نباشد، اما برای طراحان سیستم حالت تک نخی به دلیل افت کارآیی قابل قبول نیست. بنابراین روش سوم، راه اندازی سرور بعنوان یک ماشین حالت متناهی<sup>۳۲۱</sup> بزرگ است . با ورود درخواست جدید، تنها نخ موجود آنرا بررسی خواهد کرد . در صورتی که بتواند از طریق حافظه پنهان دیسک به درخواست رسیدگی کند که هیچ، و الا پیامی برای دیسک ارسال می کند.

به جای مسدود شدن، سرور وضعیت درخواست فعلی را در جدولی ثبت کرده و به سراغ پیام بعدی میرود. پیام بعدی هم ممکن است یا درخواست برای کار جدید باشد و یا پاسخی از جانب دیسک درمورد عملیات قبلی. اگر درخواست کار جدید باشد، کار آغاز خواهد شد. اگر پاسخ از دیسک باشد، اطلاعات مربوطه از جدول واکشی شده، پاسخ پردازش شده و سپس برای مشتری ارسال می گردد. در این طرح، سرور مجبور خواهد بود که از فرآخوانی های غیر مسدود شونده **send** و **receive** استفاده کند.

در این طرح، مدل "فرآیند ترتیبی" موجود در دو مورد اول دیگر بچشم نمی خورد. وضعیت محاسبه بایستی برای تمامی پیامهای ارسالی و دریافتی به صورت آشکار در جدول ذخیره و بازیابی شود. در واقع، ما نخ ها و پشته های آنها را با دشواری شبیه سازی کرده ایم. این فرآیند بصورت ماشین حالت متناهی عمل می کند، یعنی یک رویداد راگرفته و بسته به محتویات آن، در برابر آن واکنش نشان می دهد.

ویژگی ها	مدل
موازی گرایی، فرآخوانی های سیستمی مسدود شونده	چند نخ
غیر موازی ، فرآخوانی های سیستمی مسدود شونده	فرآیند تک نخی
موازی گرایی، فرآخوانی های سیستمی غیرمسدود شونده	ماشین حالت محدود

شکل ۳-۴- سه روش برای ساخت سرور

نخ ها این امکان را می دهند ایده فرآیند های ترتیبی که به وسیله فرآخوان های سیستمی مسدود شونده انجام می شوند (مثلًا **RPC** برای ارتباط با دیسک) حفظ شود و همچنان موازی سازی به دست

<sup>321</sup> Finite-State Machine

آید. فراخوان های سیستمی مسدود شونده برنامه نویسی را تسهیل کرده و موازی گرایی هم باعث ارتقاء کارآیی خواهد شد. در سرور تک نخی سهولت و سادگی فراخوانی های سیستمی مسدود شونده حفظ شده، اما کارآیی کاهش پیدا می کند. در روش ماشین حالت متناهی، موازی گرایی باعث ارتقاء کارآیی شده ولی به کار گیری فراخوان های غیر مسدود شونده برنامه نویسی را با برخی پیچیدگی ها مواجه می سازد. این مدل ها بطور خلاصه در شکل ۳-۴ ارائه شده اند.

### ۳-۲- مجازی سازی<sup>۳۲۲</sup>

نخ ها و فرآیندها را می توان بعنوان روشی برای افزایش کمیت کار در زمان یکسان تلقی نمود. در واقع، بكمک آنها می توانیم (قطعاتی از) برنامه هایی را بسازیم که ظاهراً بصورت همزمان اجرا می شوند. البته، در یک کامپیوتر تک پردازنده ای، اجرای همزمان فقط تصور می شود. از آنجا که فقط یک پردازنده وجود دارد، در هر زمان فقط یک دستور العمل از یک فرآیند یا یک نخ قابل اجراست. با سوئیچ کردن سریع بین نخ ها و فرآیندها تصور موازی گرایی ایجاد می شود.

این تفکیک بین داشتن یک پردازنده و تظاهر به وجود بیش از یکی را می توان به منابع دیگر هم بسط داده و در نتیجه به چیزی به نام **مجازی سازی منبع<sup>۳۲۳</sup>** دست یافت. هر چند مفهوم مجازی سازی از چند دهه پیش مورد استفاده قرار گرفته است، اما با افزایش کاربرد و پیچیدگی سیستم های کاربردی (توزیعی) علاقمندی جدیدی به آن مشاهده می شود، زیرا نرم افزار برنامه های کاربردی در اکثریت قریب به اتفاق موارد بیشتر از سخت افزار و نرم افزار سیستم های زیربنایی خود عمر می کنند. در این بخش، تا حدودی به نقش مجازی سازی و نحوه فعلی بخشیدن به آن خواهیم پرداخت.

### ۱-۳-۲- نقش مجازی سازی در سیستم های توزیعی

در عمل، تمامی سیستم های کامپیوتری (توزیعی)، مطابق شکل (الف) ۳-۵، واسط برنامه ریزی را برای نرم افزار سطح بالاتر ارائه می دهند. واسط ها انواع مختلفی دارند که از آن میان می توان به مجموعه دستورالعمل پایه که بوسیله پردازنده ارائه می شود تا مجموعه گسترده ای از واسط های برنامه نویسی کاربردی در بسیاری از سیستم های میان افزار امروزی اشاره کرد. بطور خلاصه و مطابق شکل (ب) ۳-۵، مجازی سازی به توسعه یا تعویض واسط فعلی با هدف تقلید رفتار سیستم دیگر می پردازد. در ادامه راجع به جزئیات فنی مجازی سازی بحث خواهیم کرد، اما اجازه دهید پیش از آن نگاهی داشته باشیم به دلیل اهمیت مجازی سازی در سیستم های توزیعی.

<sup>322</sup> virtualization

<sup>323</sup> resource virtualization

یکی از مهمترین دلایل برای معرفی مجازی سازی در سالهای دهه هفتاد ایجاد امکان اجرا شدن نرم افزار باقیمانده از گذشته<sup>۳۲۴</sup> بر روی سخت افزار کامپیوتر های بزرگ<sup>۳۲۵</sup> گران قیمت بود. این نرم افزار علاوه بر برنامه های کاربردی مختلف، شامل سیستم عامل های ایجاد شده برای آنها نیز بود. این شیوه پشتیبانی از نرم افزار باقیمانده از گذشته بطور موفقیت آمیزی در کامپیوتر های بزرگ IBM 370 (و جانشینان آنها) اجرا شده و در نتیجه ماشین مجازی بوجود آمد که سیستم عامل های مختلف به آن انتقال داده می شدند.

هم زمان با کاهش قیمت سخت افزار، کامپیوترها هم قدر تمدنتر شده، تنوع سیستم های عامل کاهش یافت و مجازی سازی دیگر مانند قبل مورد توجه نبود. با این وجود، به چند دلیل که در ادامه خواهیم گفت، از اواخر سال های دهه نود شرایط مجدداً دستخوش تغییراتی شد.

**شكل ۳-۵-الف)** سازمان عمومی بین برنامه، واسط و سیستم.**ب)** سازمان عمومی مجازی سازی سیستم A در بالای سیستم B (داخل شکل: الف) برنامه، واسط A، سخت افزار / نرم افزار سیستم A، ب) برنامه، واسط A، پیاده سازی تقلید A روی B، واسط B، سخت افزار / نرم افزار سیستم B).

اول، هرچند سخت افزار و نرم افزار سیستم های سطح پایین با سرعت زیادی تغییر می کنند، نرم افزار در سطوح بالاتر انتزاع (بعنوان مثال، میان افزار و برنامه های کاربردی)، ثبات خیلی بالاتری دارند. به بیان دیگر، نمی توان نرم افزار باقیمانده از گذشته را بر روی سکوهای گذشته نگهداری کرد. در اینجا، مجازی سازی می تواند بصورت انتقال واسط های باقیمانده از گذشته به سکوهای جدید و سپس، استفاده فوری از آنها برای گروه های بزرگی از برنامه های موجود مفید واقع شود.

یک واقعیت مهم دیگر این است که شبکه سازی کاملاً فraigir شده است. نمی توان تصور کرد که کامپیوترهای امروزی به هیچ شبکه ای متصل نباشند. در عمل، این امر مستلزم آن است که مدیران اجرایی سیستم ها مجموعه ای بزرگ و ناهمگن<sup>۳۲۶</sup> از کامپیوترهای سرور در اختیار داشته باشند که هریک برنامه های کاربردی بسیار مختلفی را اجرا کرده، و مشتری ها به راحتی بتوانند به این برنامه های کاربردی دسترسی پیدا کنند. همچنین این برنامه های کاربردی باید به منابع مختلف دسترسی داشته باشند. مجازی سازی می تواند بسیار مفید باشد، چون می تواند تنوع سکوها و ماشینها را کاهش دهد، به این ترتیب که هر یک از برنامه های کاربردی روی ماشین مجازی ویژه خود، که احتمالاً شامل کتابخانه های مربوطه و سیستم عامل می باشد، که به نوبه خود روی یک سکوی مشترک قرار دارند، اجرا می شود.

<sup>324</sup> legacy

<sup>325</sup> mainframe computers

<sup>326</sup> Heterogeneous

نوع آخر مجازی سازی جابجا پذیری<sup>۳۲۷</sup> و انعطاف پذیری بالایی را ایجاد می کند. عنوان مثال، Awadallah and Rosenblum(۲۰۰۲) بحث می کنند که برای تحقیق بخشیدن به ایده شبکه های تحويل محتوا<sup>۳۲۸</sup> که براحتی قادر به پشتیبانی از کپی برداری محتوای پویا باشند، اگر از سرورهای لبه ای<sup>۳۲۹</sup> که از مجازی سازی پشتیبانی کرده و در نتیجه برای کل سایت (شامل محیط آن) امکان کپی شدن فعالانه را فراهم آورند استفاده شود، شاهد تسهیل چشمگیر مدیریت خواهیم بود. همانطور که بعداً هم خواهیم گفت، مباحث جابجا پذیری مجازی سازی را تبدیل به یکی از مکانیزم های مهم در سیستم های توزیعی نموده است.

### ۲-۳-۲-معماری های ماشین های مجازی

برای تحقیق مجازی سازی روش های بسیار مختلفی وجود دارد. Nair و Smith (2005) مروری بر این روش ها ارائه داده اند. برای درک تفاوت ها در مجازی سازی، لازم است بدانیم که سیستم های کامپیوتری غالباً چهار نوع واسط مختلف در چهار سطح مختلف ارائه می دهند:

۱. یک واسط بین سخت افزار و نرم افزار، شامل دستورالعمل های ماشین که از طریق هر

برنامه ای قابل اجرا است.

۲. یک واسط بین سخت افزار و نرم افزار ، شامل دستورالعمل های ماشین که فقط توسط برنامه های ممتاز<sup>۳۳۰</sup> از قبیل سیستم عامل قابل اجرا است.

۳. یک واسط شامل فراخوان های سیستمی که توسط سیستم عامل ارائه می شود.

۴. یک واسط شامل فراخوان های کتابخانه ای که غالباً باعث ایجاد چیزی به نام واسط برنامه نویسی کاربردی<sup>۳۳۱</sup>(API) می شود. در بسیاری از موارد، فراخوان های

سیستمی بوسیله یک API مخفی می شوند.

این انواع مختلف در شکل ۳-۶ نمایش داده شده است . بطور خلاصه، هدف از مجازی سازی تقلید رفتار این واسط ها است.

شکل ۳-۶- واسط های مختلف ارائه شده توسط سیستم های کامپیوتری( برنامه کاربردی، توابع کتابخانه، کتابخانه، فراخوان های سیستمی، سیستم عامل، دستورالعمل های ممتاز، دستورالعمل های عمومی، سخت افزار).

مجازی سازی ممکن است به دو روش مختلف انجام شود. اولاً می توانیم یک سیستم زمان اجرا<sup>۳۳۲</sup> بسازیم که اساساً مجموعه دستورالعمل انتزاعی را برای اجرای برنامه های کاربردی ارائه دهد. این

<sup>327</sup> Portability

<sup>328</sup> Content Delivery Networks

<sup>329</sup> Edge Servers

<sup>330</sup> Privileged

<sup>331</sup> Application Programming Interface

<sup>332</sup> Runtime System

دستورالعمل ها می توانند ترجمه شوند (مانند محیط زمان اجرای Java)، همچنین می توانند تقليید شوند مشابه کاری که در اجرای برنامه های کاربردی ویندوز روی سکوهای UNIX انجام می شود. توجه داشته باشید که در این حالت، مقلد<sup>۳۳۳</sup> مجبور خواهد بود تا رفتار فراخوان های سیستمی را نیز تقليید کند که در عمل چندان هم ساده نیست. اين نوع مجازی سازی منجر به چیزی می شود که Nair و Smith (۲۰۰۵) از آن با عنوان **ماشین مجازی فرآیند**<sup>۳۳۴</sup> یاد کرده اند و تأکیدی است بر این واقعیت که مجازی سازی اساساً فقط برای یک فرآیند انجام می شود.

یک روش دیگر برای مجازی سازی، ایجاد سیستمی است که اساساً بعنوان لایه ای پیاده سازی شده است که سخت افزار اصلی را می پوشاند و مجموعه دستورالعمل کاملی از همان نوع ( یا سخت افزار دیگر) را بعنوان واسطه ارائه می کند. نکته مهم آن است که این واسط را می توان بصورت همزمان به برنامه های مختلف ارائه کرد. در اینصورت ، امکان اجرای مستقل و همزمان چندین سیستم عامل مختلف روی یک سکو واحد فراهم خواهد آمد. این لایه غالباً به نام **ناظر ماشین مجازی**<sup>۳۳۵</sup>(VMM) خوانده می شود. از نمونه های این روش می توان به VMware (۲۰۰۱)، Sugerman و گروه Barham (۲۰۰۳) و Xen ( ۲۰۰۷) همکاران) اشاره کرد. این دو رویکرد مختلف در شکل ۳ نمایش داده شده است.

شکل ۳-۷ - الف) یک ماشین مجازی فرآیند همراه با چند نمونه از ترکیبات ( برنامه کاربردی، سیستم زمان اجرا). ب) ناظر ماشین مجازی همراه با چند نمونه های از ترکیبات ( برنامه های کاربردی، سیستم عامل) (داخل شکل: الف) برنامه کاربردی، سیستم زمان اجرا، سیستم عامل، سخت افزار، ب) برنامه های کاربردی، سیستم عامل، ناظر ماشین مجازی، سخت افزار).

همانطور که توسط (۲۰۰۵) Rosenblum و Garfinkel VMM ها در بافت قابلیت وثوق و امنیت سیستم های توزیعی) به صورت روزافزونی افزایش خواهد یافت. همچنین، از آنجایی که VMM ها باعث تفکیک کل برنامه کاربردی از محیط آن می شوند، خرابی ناشی از خطا یا حملات امنیتی دیگر بر کل ماشین تأثیرگذار نخواهد بود. بعلاوه، همانطور که قبلًا هم گفتیم، VMM ها با فراهم آوردن امکان تفکیک بیشتر بین سخت افزار و نرم افزار و امكان انتقال کامل محیط از یک ماشین به ماشین دیگر، باعث افزایش چشمگیر جابجا پذیری می شوند.

<sup>333</sup> Emulator

<sup>334</sup> Process Virtual Machine

<sup>335</sup> virtual machine monitor

### ۳-۳- مشتری ها

در فصلهای قبل راجع به مدل مشتری-سرور ، نقش های مشتری ها و سرورها و روشهای تعامل آنها صحبت کردیم. حال بباید نگاه دقیقتی به ساختار مشتری و سرور داشته باشیم . این بخش را با بحث در مورد مشتری ها آغاز کرده و سرورها را به بخش بعد موکول می کنیم.

#### ۱-۳-۳- واسط های کاربر شبکه ای شده<sup>۳۳۶</sup>

یکی از مهمترین وظایف ماشین های مشتری، ایجاد ابزار لازم برای تعامل کاربران با سرورهای دور است. برای پشتیبانی از این تعامل دو راه وجود دارد: اولاً ، به ازای هر سرویس از راه دور، ماشین مشتری یک نظری جدایانه داشته باشد تا بتواند روی شبکه با سرویس مورد نظر تماس برقرار نماید. بعنوان نمونه می توان به دستور کاری<sup>۳۳۷</sup> اشاره کرد که روی PDA کاربر اجرا شده و بایستی با یک دستور کار دور، و احتمالاً اشتراکی دیگر همگام شود. در این حالت و مطابق شکل (الف) ۳-۸، یک پروتکل در سطح برنامه کاربردی<sup>۳۳۸</sup> کار همگام سازی را انجام خواهد داد.

شکل ۳-۸- (الف) یک برنامه کاربردی شبکه ای با پروتکل خود. (ب) یک راه حل کلی برای ایجاد دسترسی به برنامه های کاربردی دور (داخل شکل: (الف) ماشین مشتری، برنامه کاربردی، میان افزار، سیستم عامل محلی، پروتکل ویژه برنامه کاربردی، ماشین سرور، برنامه کاربردی، میان افزار، سیستم عامل محلی، شبکه، (ب) ماشین مشتری، برنامه کاربردی، میان افزار، سیستم عامل محلی، پروتکل مستقل از برنامه کاربردی، ماشین سرور، برنامه کاربردی، میان افزار، سیستم عامل محلی، شبکه).

راه حل دوم، ایجاد دسترسی مستقیم به سرویسه ای دور صرفاً با استفاده از یک واسط کارآمد است. در عمل این بدان معناست که ماشین مشتری صرفاً بعنوان پایانه ای بدون نیاز به ذخیره سازی محلی بکار بردشده و در نتیجه، راه حل مستقل از برنامه کاربردی مطابق شکل (ب) ۳-۸ بوجود خواهد آمد. در واسط های کاربر شبکه ای همه چیز در طرف سرور پردازش و ذخیره می شود. همراه با افزایش امکان اتصال به اینترنت و پیچیده تر شدن وسایل دستی، رویکرد روش مشتری لاغر<sup>۳۳۹</sup> بیش از پیش مورد توجه قرار گرفته است. همانطور که در فصل قبل هم گفتیم، دلیل دیگر برای کاربرد فراوان راه حل های مشتری لاغر آن است که باعث تسهیل مدیریت سیستم می شوند. در ادامه نگاهی خواهیم داشت به نحوه پشتیبانی از واسط های کاربر شبکه ای.

<sup>336</sup> networked user interfaces

<sup>337</sup> agenda

<sup>338</sup> application-level protocol

<sup>339</sup> thin-client approach

## مثال: سیستم X Window

احتمالاً یکی از قدیمی ترین و درعین حال پرکاربردترین واسطه های کاربر شبکه ای شده، سیستم **X Window** می باشد. سیستم **X Window** که غالباً فقط به نام **X** خوانده می شود، برای کنترل پایانه های نگاشت بیتی<sup>۳۴۰</sup> بکار می رود که شامل یک صفحه نمایش، صفحه کلید و ابزار اشاره مانند ماوس است. از یک نظر، **X** را می توان بخشی از سیستم عامل تلقی نمود که پایانه را کنترل می کند. قلب سیستم از چیزی به نام **هسته اصلیX**<sup>۳۴۱</sup> تشکیل شده که شامل تمام درایورهای وسائل ترمینال بوده که معمولاً به شدت به سخت افزار وابسته می باشند.

هسته اصلی **X** واسطه نسبتاً سطح پایینی را برای کنترل صفحه نمایش و همچنین گرفتن رویدادها از صفحه کلید و ماوس ارائه می دهد. مطابق شکل ۳-۹، این واسطه بصورت کتابخانه ای به نام **Xlib** در اختیار برنامه های کاربردی قرار می گیرد.

شكل ۳-۹-سازمان اصلی سیستم **X Window** (داخل شکل: سرور برنامه کاربردی، مدیر پنجره، **Xlib** سیستم عامل محلی، سرور برنامه کاربردی، برنامه کاربردی، واسط **Xlib** سیستم عامل محلی، پروتکل **X** پایانه کاربر، هسته اصلی **X** ، درایور های دستگاه، پایانه (شامل صفحه نمایش، صفحه کلید، ماوس و غیره).

یکی از نکات جالب توجه در مورد **X** آن است که هسته اصلی **X** و برنامه های کاربردی **X** نباید لزوماً روی یک ماشین واحد مقیم باشند. خصوصاً اینکه ، **X** پروتکل ارتباطی سطح برنامه کاربردی به نام **پروتکل X**<sup>۳۴۲</sup> ارائه می دهد که بوسیله یک مورد از **Xlib** قادر به تبادل داده ها و رویدادها با هسته اصلی **X** خواهد بود. بعنوان مثال، **Xlib** قادر است تا برای ایجاد یا حذف پنجره ، تنظیم رنگها و تعریف نوع شاخص صفحه نمایش و بسیاری از درخواست های دیگر، اقدام به ارسال درخواست به هسته اصلی **X** نماید. هسته اصلی **X** هم متقابلاً با بازگرداندن بسته های رویداد به **Xlib** ، در برابر واقعیع محلی از قبیل ورودی ماوس و صفحه کلید واکنش نشان خواهد داد.

امکان ارتباط همزمان چندین برنامه کاربردی با هسته اصلی **X** فراهم شده است. یکی از برنامه های کاربردی بنام **مدیر پنجره**<sup>۳۴۳</sup> حق ویژه ای در این رابطه دارد. این برنامه کاربردی قادر است تا "اظاهر و حس" صفحه نمایش را به همان صورتی که به نظر کاربر می رسد، مشخص کند. بعنوان مثال، مدیر پنجره می تواند تعیین کند که هریک از پنجره ها با استفاده از دکمه های اضافی چگونه ترئین شوند، پنجره ها چگونه روی صفحه نمایش قرار گیرند وغیره. دیگر برنامه های کاربردی هم باید از این قوانین تبعیت کنند.

<sup>340</sup> bit-mapped terminals

<sup>341</sup> X kernel

<sup>342</sup> X protocol

<sup>343</sup> window manager

بهتر است درمورد تناسب عملی سیستم  $X$  در سیستم مشتری-سرور هم نکته ای را بیان کنیم. از آنچه تا به اینجا گفته شد می‌توان نتیجه گرفت که هسته اصلی  $X$  درخواست هایی را برای دستکاری<sup>۳۴۴</sup> صفحه نمایش دریافت می‌کند. این درخواست‌ها از برنامه‌های کاربردی (احتمالاً دور) دریافت می‌شوند. در این حالت، هسته اصلی  $X$  بعنوان سرور عمل خواهد کرد، در حالی که برنامه‌های کاربردی در نقش مشتری ظاهر می‌شوند. از این واژه گزینی در  $X$  استفاده می‌شود، و اگرچه صحیح است ولی ممکن است باعث ابهام شود.

### مشتری لاغر در کاربردهای شبکه ای

برنامه‌های کاربردی بكمک فرامين خاص صفحه نمایش که توسط  $X$  ارائه می‌شود، صفحه نمایش را دستکاری می‌کنند. اين فرامين غالباً از طریق شبکه ارسال و سپس به وسیله هسته اصلی  $X$  اجرا می‌شوند. برنامه‌های کاربردی نوشته شده برای  $X$  باید منطق برنامه کاربردی را از فرامين واسط کاربر تفکیک کنند. متأسفانه غالباً اين حالت اتفاق نمی‌افتد. در نتیجه، همانطور که (۲۰۰۲) Lai and Nieh هم گزارش داده اند، قسمت عمدۀ منطق برنامه کاربردی و تعامل کاربر ارتباط محکمی با هم دارند؛ به این معنی که برنامه کاربردی درخواست‌های زیادی را برای هسته اصلی  $X$  ارسال کرده و در عوض انتظار دارد تا پیش از اقدام به مرحله بعد، پاسخی از جانب آن دریافت کند. این نوع رفتار همگام، در صورت اعمال روی یک شبکه حوزه گسترده با تأخیرهای طولانی، ممکن است تأثیرات زیادی بر عملکرد داشته باشد.

راه حل‌های مختلفی برای این مشکل وجود دارد. یک روش، طراحی مجدد پیاده سازی پروتکل  $X$  است، همان گونه که در NX (Pinzari, ۲۰۰۳) انجام شده است. یک بخش مهم این کار، کاهش پهنای باند از طریق متراکم سازی پیامهای  $X$  است. در ابتدا فرض می‌شود که پیام‌ها از یک بخش ثابت، که بعنوان شناسه تلقی می‌شود، و یک بخش متغیر تشکیل شده اند. در بسیاری از موارد، چندین پیام یک شناسه واحد دارند که در اینصورت، غالباً حاوی داده‌های مشابهی می‌باشند. از این ویژگی می‌توان فقط برای ارسال تفاوت‌ها بین پیامهای دارای شناسه واحد استفاده کرد.

هر دو طرف فرستنده و گیرنده دارای حافظه پنهان محلی هستند که می‌توان با استفاده از شناسه پیام، اقدام به جستجو در درایه<sup>۳۴۵</sup> های آن نمود. پس از ارسال پیام، جستجو ابتدا در حافظه پنهان محلی انجام می‌شود. اگر در آنجا یافت شود، به این معناست که قبلًاً پیامی با همان شناسه، اما با داده‌های احتمالاً متفاوت، ارسال شده بود. در این حالت، برای ارسال صرفاً تفاوت‌های بین ایندو از کدگذاری تفضیلی<sup>۳۴۶</sup> استفاده می‌شود. در طرف دریافت کننده، پیام در حافظه پنهان محلی هم جستجو می‌شود و سپس، رمزگشایی از طریق تفاوت‌ها صورت می‌پذیرد. در صورت یافت نشدن در حافظه مخفی، از

<sup>344</sup> Manipulate

<sup>345</sup> Entry

<sup>346</sup> differential encoding

روشهای متراکم سازی استانداردی استفاده می شود که غالباً باعث بهبود ضریب چهاری پهنای باند می شود. در مجموع، گزارش شده که این تکنیک منجر به کاهش پهنای باند تا ۱۰۰۰ برابر شده و در نتیجه، X از طریق پیوندهای با پهنای باند کم تا ۹۶۰۰ kbps هم قابل اجرا خواهد بود. یکی از اثرات جانبی مهم ذخیره پیام ها در حافظه پنهان آن است که فرستنده و گیرنده اطلاعات مشترکی درمورد وضعیت فعلی صفحه نمایش خواهند داشت. بعنوان مثال، یک برنامه کاربردی می تواند صرفاً با درخواست جستجوها در حافظه پنهان محلی ، اطلاعات هندسی درباره اشیاء مختلف درخواست نماید. با در اختیار داشتن صرفاً همین اطلاعات اشتراکی، تعداد پیامهای لازم جهت همگام نگه داشتن برنامه کاربردی و صفحه نمایش کاهش خواهد یافت.

علیرغم تمام این تغییرات، هنوز هم X نیازمند داشتن یک سرور صفحه نمایش در حال اجراست. این امر ممکن است زیاد به نظر برسد، خصوصاً در مواردی که صفحه نمایش یک تلفن همراه ساده باشد. یک راه حل برای حفظ سادگی نرم افزار در صفحه نمایش آن است که تمام پردازش ها در طرف برنامه کاربردی انجام شود. در عمل این بدان معناست که کل صفحه نمایش تا سطح پیکسل در طرف برنامه کاربردی کنترل می شود. سپس تغییرات نگاشت بیتی از طریق شبکه به صفحه نمایش ارسال و از آنجا فوراً به حافظه میانگیر فریم محلی <sup>۳۴۷</sup> انتقال داده می شوند.

این رویکرد مستلزم استفاده از تکنیکهای فشرده سازی پیشرفته است تا بتواند مانع شود که در دسترس بودن پهنای باند به مشکل تبدیل شود. به عنوان مثال نمایش یک جریان ویدئوی با سرعت ۳۰ فریم در هر ثانیه روی صفحه نمایشی با ابعاد  $320 \times 240$  را درنظر بگیرید. صفحه نمایشی با این ابعاد در بسیاری از PDAها به چشم می خورد. چنانچه هریک از پیکسلها با ۲۴ بیت کدگذاری شود، آنگاه بدون فشرده سازی نیازمند پهنای باندی به اندازه تقریبی ۵۳ Mbps خواهیم بود. ناگفته پیداست که در این حالت فشرده سازی ضروری است و در حال حاضر از تکنیکهای زیادی برای اینکار استفاده می شود. اما توجه داشته باشید که فشرده سازی مستلزم بازسازی در طرف مشتری است و این ممکن است در صورت عدم حمایت سخت افزاری، هزینه های محاسباتی بالایی داشته باشد. پشتیبانی سخت افزاری می تواند مهیا شود، اما باعث افزایش قیمت آن دستگاه خواهد شد.

نقطه ضعف ارسال داده های خام پیکسل در قیاس با پروتکلهای سطح بالاتری از قبیل X آن است که معنی شناسی <sup>۳۴۸</sup> های برنامه کاربردی در سطح ارسال داده های خام از دست می روند و امکان استفاده از آنها وجود ندارد. Baratto (۲۰۰۵) و گروه همکاران راه حل متفاوتی را پیشنهاد کرده اند. در این روش که THINC نام دارد، تعداد محدودی فرمان نمایش سطح بالا ایجاد کرده اند که در سطح درایورهای دستگاه ویدئو عمل می کنند. این فرامین، وابسته به دستگاه هستند و قادر ترندتر از عملیات داده های خام خواهند بود، و در عین حال در قیاس با آنچه پروتکلی از قبیل X ارائه می دهد، ضعیف

<sup>347</sup> local frame buffer

<sup>348</sup> Semantics

تر هستند. نتیجه اینکه سورورهای نمایش می توانند بسیار ساده تر شوند که برای میزان به کارگیری پردازنده خوب است. در عین حال، می توان از بهینه سازی های وابسته به برنامه کاربردی جهت کاهش پهنای باند و همگام سازی استفاده نمود.

در THINC، درخواستهای صفحه نمایش از برنامه کاربردی گرفته شده و به فرامین سطح پایین تر ترجمه می شوند. بكمک گرفتن درخواست های برنامه های کاربردی، THINC قادر است تا از معنی شناسی های برنامه کاربردی جهت تصمیم گیری در مورد بهترین صورت ترکیبی فرامین سطح پایین تر استفاده کند. فرامین ترجمه شده فوراً به صفحه نمایش ارسال نشده، بلکه در عوض صف بندی می شوند. بسته بندی<sup>۳۴۹</sup> چندین فرمان مختلف باعث متراکم سازی فرامین صفحه نمایش به شکل یک فرمان واحد و در نتیجه کاهش تعداد پیام ها خواهد شد. بعنوان مثال، اگر پیام جدیدی در مورد نقاشی کردن در منطقه خاصی از صفحه نمایش عملاً برچیزی که فرمان (هنوز صفتندی شده) قبلی ایجاد کرده رونویسی شود، دیگر نیازی به ارسال فرمان قبلی به صفحه نمایش وجود نخواهد داشت. نهایتاً اینکه، به جای اینکه صفحه نمایش درخواست به هنگام سازی داشته باشد، THINC همواره به هنگام سازی ها را به همان ترتیب آماده شدن ارسال خواهد کرد (هل می دهد). بعلاوه، چون دیگر لزومی برای ارسال درخواست به هنگام سازی توسط صفحه نمایش وجود ندارد، روش هل دادن باعث کاهش تأخیر خواهد شد.

در نتیجه، روش مورد استفاده در THINC باعث افزایش عملکردکلی می شود، و هماهنگی بسیار زیادی با NX دارد. جزئیات بیشتر در مورد مقایسه عملکردها را در (۲۰۰۵) Baratto و گروه همکاران مطالعه کنید.

### اسناد ترکیبی<sup>۳۵۰</sup>

واسطه های کاربر امروزی بسیار بیشتر از آن چه به توسط سیستم هایی از قبیل X یا برنامه های کاربردی ساده آن انجام می شود کار می کنند. به خصوص، بسیاری از واسطه های کاربر به برنامه های کاربردی امکان می دهند تا در یک پنجره گرافیکی واحد شریک شده و از طریق عملیات کاربر، از این پنجره جهت تبادل داده ها استفاده کنند. از دیگر کارهای قابل انجام توسط کاربر می توان به ترتیب به عملیات کشیدن و رها کردن<sup>۳۵۱</sup> و ویرایش در محل<sup>۳۵۲</sup> اشاره کرد.

بعنوان یکی از نمونه های کاربرد کشیدن و رها کردن می توان به انتقال آیکن نماینده فایل A به آیکن نماینده سطل آشغال و حذف آن اشاره کرد. در این حالت، واسط کاربر باید کاری بیش از صرفاً مرتب سازی آیکنهای روی صفحه نمایش انجام دهد؛ یعنی بایستی به محض انتقال آیکن A به بالای آیکن

<sup>349</sup> batching

<sup>350</sup> compound documents

<sup>351</sup> drag-and-drop

<sup>352</sup> in-place editing

برنامه کاربردی سطل آشغال ، نام فایل A را به برنامه کاربردی مربوط به سطل آشغال انتقال دهد. به راحتی می توانید مثال های دیگری در این رابطه ذکر کنید.

ویرایش در محل را می توان به بهترین نحو در اسنادی که شامل متن و تصاویر گرافیکی هستند مشاهده کرد. فرض کنید که سند در داخل یک پردازشگر کلمه استاندارد نمایش داده شده باشد. به محض اینکه کاربر موس را در بالای شکل قرار می دهد، واسط کاربر اطلاعات آن را به برنامه نقاشی می دهد تا کاربر بتواند شکل در داخل سند تأثیر خواهد گذاشت . بنابراین واسط کاربر ارتفاع و عرض شکل جدید را محاسبه کرده و این اطلاعات را به پردازشکر کلمه می دهد. به این ترتیب، پردازشگر قادر به هنگام سازی اتوماتیک آرایش صفحه سند خواهد شد.

ایده اصلی در واسط های کاربر همان ایده سند ترکیبی است که بصورت مجموعه ای از اسناد احتمالاً بسیار متفاوت ( از قبیل متن، شکل، صفحه گسترده و غیره) تعریف می شود که بصورتی روان در سطح کاربر- واسط ادغام شده اند. واسط کاربری که می تواند با سندهای ترکیبی کار کند این واقعیت را که برنامه های کاربردی مختلف روی بخشهاي مختلف سند کار می کنند را مشخص می کند. از نظر کاربر ، تمامی بخشها به صورتی روان با هم تجمعی شده اند. چنانچه تغییر یک بخش بر بخشهاي دیگر هم تأثیر بگذارد، واسط کاربر می تواند کار مناسب را انجام دهد، مثل مطلع ساختن برنامه های کاربردی. مشابه آنچه در مورد سیستم Window X گفته شد، لزومی ندارد برنامه های کاربردی مربوط به یک سند ترکیبی روی ماشین مشتری اجرا شوند. با این وجود، باید بدانیم که واسط های کاربری که از اسناد ترکیبی پشتیبانی می کنند، ممکن است کار پردازش بیشتری نسبت به دیگرانی که فاقد این مزیت هستند انجام دهند.

### ۳-۳-۲- نرم افزار طرف مشتری برای شفافیت توزیعی<sup>۳۵۳</sup>

نرم افزار مشتری از چیزی بیش از صرفاً واسط های کاربر تشکیل شده است. در بسیاری از موارد، بخشهايی از سطح پردازش و سطح داده برنامه کاربردی مشتری- سرور نیز در طرف مشتری اجرا می شوند. ماشین های پاسخ گوی اتوماتیک ATM<sup>۳۵۴</sup> (ها)، ماشین های فروش نقدی<sup>۳۵۵</sup> ، دستگاه های خواننده بارکد، دستگاه های روی تلویزیون<sup>۳۵۶</sup> و غیره است. در این موارد دیده می شود واسط کاربر بخش نسبتاً کوچکی از نرم افزار مشتری است، برخلاف تجهیزات ارتباطی و پردازش محلی.

نرم افزار مشتری ، علاوه بر واسط کاربر و نرم افزارهای مربوط به برنامه کاربردی، متشکل از مؤلفه هایی برای کسب شفافیت توزیعی است. ایده آل آن است که مشتری از ارتباط خود با فرآیندهای راه دور اطلاع نداشته باشد. در حالی که به دلایل کارآیی و درستی، توزیع غالباً برای سرورها شفافیت کمتری

<sup>353</sup> distribution transparency

<sup>354</sup> Automati Teller Machine

<sup>355</sup> Cash Register

<sup>356</sup> TV Set-Top Box

دارد. بعنوان مثال، در فصل ۶ خواهیم گفت که در مواردی لازم است سرورهای کپی شده با هم ارتباط برقرار کنند تا عملیات با ترتیب خاص در تمامی نسخه های کپی اجرا شوند.

شفافیت دسترسی معمولاً با تولید بخش باقیمانده مشتری<sup>۳۵۷</sup> از تعریف واسطه<sup>۳۵۸</sup> برای آنچه سرور باید ارائه دهد، ایجاد می شود. این باقیمانده همان واسطه موجود برای سرور را ارائه می دهد، اما تفاوت های احتمالی در معماری های ماشین و همچنین ارتباطات واقعی را مخفی می کند.

روشهای مختلفی برای اجرای شفافیت مکانی، مهاجرتی، و تغییر مکانی وجود دارد. همانطور که در فصل بعد هم خواهیم دید، استفاده از یک سیستم نامگذاری مناسب ضرورت دارد. در بسیاری از موارد، همکاری با نرم افزار طرف مشتری هم اهمیت پیدا می کند. بعنوان مثال، در حالی که مشتری به سرور بسته شده است، می توان تغییر محل سرور را مستقیماً به مشتری اطلاع داد. در این حالت، میان افزار مشتری می تواند محل استقرار جغرافیایی فعلی سرور را از کاربر مخفی نموده و در صورت لزوم، به صورتی شفاف مجدداً به سرور بسته شود. در بدترین حالت، برنامه کاربردی مشتری ممکن است متوجه افت کارآیی وقت شود.

به صورتی مشابه، بسیاری از سیستم های توزیعی به کمک راه حل های طرف مشتری اقدام به اجرای شفافیت نسخه برداری می نمایند. بعنوان مثال، یک سیستم توزیعی مجهز به سرورهای نسخه برداری شده را در نظر بگیرید. مطابق شکل ۱۰ - ۳، این نسخه برداری می تواند از طریق ارسال درخواست برای هریک از نسخه ها به انجام برسد. نرم افزار طرف مشتری می تواند بصورتی شفاف تمامی پاسخها را جمع آوری کرده و یک پاسخ واحد را به برنامه کاربردی طرف مشتری انتقال دهد.

شکل ۳-۱۰ - نسخه برداری شفاف از سرور با استفاده از راه حل طرف مشتری (داخل شکل: ماشین مشتری، برنامه کاربردی مشتری، طرف مشتری درخواست چندگانه را اداره می کند. سرور۱، برنامه کاربردی سرور، درخواست نسخه برداری شده، سرور۲، برنامه کاربردی سرور، سرور۳، برنامه کاربردی سرور).

در آخر هم نوبت به شفافیت خرابی<sup>۳۵۹</sup> می رسد. مخفی سازی خرابی های ارتباطی با یک سرور نوعاً توسط میان افزار مشتری انجام می شود. بعنوان مثال، میان افزار مشتری را می توان به صورتی پیکره بندی کرد که به صورت مکرر برای اتصال به یک سرور تلاش کرده و یا این که پس از چندین تلاش ناموفق، سرور دیگری را آزمایش کند. حتی در برخی موارد، میان افزار مشتری داده هایی را که طی جلسه قبل در حافظه پنهان قرار داده بود، بازمی گرداند - گاهی اوقات جستجوگرهای وب که قادر به اتصال به سرور نمی شوند، به این صورت عمل می کنند.

<sup>357</sup> Client Stub

<sup>358</sup> Interface Definition

<sup>359</sup> Failure

شفافیت همروندی<sup>۳۶۰</sup> را می توان بكمک برخی سرورهای میانی، مثلًا مانیتورهای تراکنش، با نیاز کمتر به پشتیبانی از جانب نرم افزار مشتری اجرا نمود . شفافیت ماندگاری<sup>۳۶۱</sup> هم در اغلب موارد بطور کامل در طرف سرور انجام می شود.

### ۴-۳- سرورها

حال ببایید نگاه دقیقتری به سازمان سرورها داشته باشیم . در صفحات بعد، ابتدا روی چند مسأله کلی در طراحی سرورها تمرکز کرده و سپس به سراغ مبحث خوشه های سرور<sup>۳۶۲</sup> می رویم.

#### ۱- ۳-۴- مسائل کلی مربوط به طراحی ها

سرور یک فرآیند اجراکننده یک سرویس خاص برای مجموعه ای از مشتری هاست. در واقع هر سرور از سازمان واحدی تبعیت می کند : منتظر ورورد درخواست از جانب مشتری می ماند، به درخواست ورودی رسیدگی کرده و سپس مجدداً منتظر ورود درخواست بعدی می شود.

روشهای مختلفی برای سازماندهی سرورها وجود دارد. در مورد سرور تکراری<sup>۳۶۳</sup> ، خود سرور به درخواست رسیدگی کرده و، در صورت لزوم، پاسخی را به مشتری درخواست کننده عودت می دهد. در حالیکه سرور همرونده<sup>۳۶۴</sup> روی درخواست عمل نمی کند، بلکه آنرا به یک نخ جداگانه یا فرآیند دیگر رد کرده و سپس به سرعت منتظر ورورد درخواست بعدی می شود. سرور چندنخی نمونه ای از یک سرور همرونده می باشد. یک پیاده سازی جایگزین برای سرور همرونده می تواند ایجاد فرآیند جدید به ازای هر درخواست جدید اشاره کرد. از این روش در بسیاری از سیستم های UNIX استفاده می شود. نخ یا فرآیندی که روی در خواست کار می کند مسئول بازگرداندن درخواست به مشتری درخواست کننده هم می باشد.

مسأله دیگر محل برقراری تماس توسط مشتریان با سرور است. در تمام موارد، مشتریان درخواستها را برای یک نقطه پایانی<sup>۳۶۵</sup> ، یا به عبارت دیگر یک پایانه<sup>۳۶۶</sup> ، روی ماشین اجرا کننده سرور ارسال می کنند. هر سرور به نقطه پایانی مشخصی گوش می دهد. مشتریان چگونه نقطه پایانی یک سرویس را شناسایی می کنند؟ یک روش برای اینکار، تخصیص دادن نقاط پایانی جهانی برای سرویسهای معروف است. بعنوان مثال، سرورهایی که روی درخواستهای FTP اینترنت کار می کنند، همواره به پایانه ۲۱ TCP و سرور HTTP شبکه جهانی وب همیشه به پایانه ۸۰ TCP گوش می دهد . این نقاط پایانی

<sup>360</sup> Concurrency

<sup>361</sup> Persistency

<sup>362</sup> Server Clusters

<sup>363</sup> iterative server

<sup>364</sup> concurrent server

<sup>365</sup> end point

<sup>366</sup> port

به وسیله بخش IANA<sup>۳۶۷</sup> تخصیص یافته و اسناد آن در مرجع (۱۹۹۴) Reynolds and Postel ارائه شده است. با تخصیص نقاط پایانی ، مشتری فقط باید آدرس شبکه ای ماشینی را پیدا کند که سرور در آن در حال اجراست. همانطور که در فصل بعد هم خواهیم گفت، برای این منظور می توان از سرویس های نامگذاری استفاده نمود.

بسیاری از سرویس ها نیازی به نقاط پایانی از پیش تخصیص یافته ندارند. بعنوان مثال، یک سرور زمان روز<sup>۳۶۸</sup> ممکن است از نقطه پایانی استفاده کند که به صورت پویا و به توسط عامل محلی به آن نسبت داده شده است. در این حالت، ابتدا مشتری باید نقطه پایانی را شناسایی کند. یک راه حل برای اینکار، اجرا کردن فرآیند محافظت<sup>۳۶۹</sup> روی هریک از ماشینهای اجراکننده سرورهاست. این فرآیند محافظت نقطه پایانی فعلی تمامی سرویس های پیاده سازی شده در این ماشین را پی گیری کرده و در اختیار دارد. خود فرآیند محافظت هم به یک نقطه پایانی معروف گوش می دهد. مطابق شکل (الف) ۱۱-۳، مشتری ابتدا با فرآیند محافظت تماس برقرار نموده ، نقطه پایانی را درخواست و سپس با سرور مورد نظر تماس برقرار می کند.

شكل ۱۱-۳-الف) پیوند مشتری به سرور با استفاده از فرآیند محافظت.ب) پیوند مشتری به سرور با استفاده از ابرسرور (داخل شکل: الف) ماشین مشتری، مشتری، ۲-درخواست سرویس، ۱-درخواست نقطه پایانی، ماشین سرور، سرور، فرآیند محافظت، ثبت نقطه پایانی، جدول نقطه پایانی، ب) ماشین مشتری، مشتری، ۲-ادامه سرویس، ۱-درخواست سرویس، ماشین سرور، سرور، ایجاد سرور برای سرویس درخواست شده).

معمولًا به هر سرور یک نقطه پایانی نسبت داده می شود. با این وجود، اجرای واقعی هریک از سرویسها بوسیله یک سرور جداگانه ممکن است نوعی اتلاف منابع محسوب شود. بعنوان مثال ، در یک سیستم UNIX ، معمولًا با تعداد زیادی سرور مواجه هستیم که بطور همزمان اجرا شده و اغلب آنها تا زمان رسیدن درخواست جدید از جانب مشتری در حال انتظار و غیرفعال باقی می مانند. به جای پی گیری چندین فرآیند غیرفعال ، مطابق شکل ب) ۱۱-۳، در اغلب موارد بهتر است که یک ابرسرور<sup>۳۷۰</sup> به تمامی نقاط پایانی مربوط به سرورهای مشخصی گوش دهد. از این روش بعنوان مثال در فرآیند محافظت UNIX inetd در inetd استفاده شده است. به تعدادی از پایانه های معروف سرویس های اینترنت گوش می دهد. با ورورد درخواست جدید، فرآیند محافظت فرآیند جدیدی را برای رسیدگی به درخواست ورودی ایجاد می کند. این فرآیند پس از تکمیل سرویس خارج خواهد شد.

<sup>367</sup> Internet Assigned Numbers Authority

<sup>368</sup> time-of-day

<sup>369</sup> daemon

<sup>370</sup> superserver

مسئله دیگری که باید در طراحی سرورها لاحظ شود، امکان و چگونگی وقفه<sup>۳۷۱</sup> سرورهاست . بعنوان مثال، کاربری را در نظر بگیرید که قصد دارد فایل بسیار بزرگی را به یک سرور FTP آپلود کند . اما ناگهان متوجه می شود که فایل اشتباه بوده و تصمیم می گیرد سرور را متوقف کند تا تبادل داده های بعدی انجام نشود. برای اینکار روش های مختلفی وجود دارد. یکی از روش هایی که کار کرد بسیار خوبی در دنیای اینترنت امروز پیدا کرده ( و در بعضی موارد تنها گزینه هم محسوب می شود) این است که کاربر بطور ناگهانی از برنامه کاربردی مشتری خارج شده ( اینکار باعث شکست اتوماتیک اتصال به سرور خواهد شد)، بی درنگ آنرا مجدداً راه اندازی نماید، و تظاهر کند که هیچ اتفاقی نیفتاده. سرور هم با این تصور که مشتری احتمالاً خراب شده است، نهایتاً اتصال قدیمی را می شکند.

یک روش بسیار بهتر برای ایجاد وقفه در ارتباطات آن است که سرور و مشتری بگونه ای ایجاد شوند که امکان ارسال داده های خارج از باند<sup>۳۷۲</sup> - یعنی داده هایی که باید پیش از ورود داده های دیگر از همان مشتری، پردازش شوند- فراهم آید . یک راه حل این است که سرور به نقطه پایانی کنترل جداگانه ای گوش دهد که داده های خارج از باند توسط مشتری به آن ارسال می شود و در عین حال ( با اولویت کمتر) به نقطه پایانی دیگری گوش دهد که داده های عادی از طریق آن عبور داده می شوند. راه حل دیگر ، ارسال داده های خارج از باند از طریق همان اتصالی است که مشتری بوسیله آن درخواست اصلی را ارسال نموده است. بعنوان مثال، در TCP، می توان داده های فوریتی<sup>۳۷۳</sup> را ارسال نمود. سرور پس از دریافت داده های فوریتی ، متوقف شده ( بعنوان مثال در سیستم های UNIX اینکار توسط یک سیگنال انجام می شود) و سپس داده ها را بررسی نموده و به نحو مقتضی روی آنها عمل کند.

آخرین و مهمترین نکته ای که باید در طراحی ها مدنظر قرار گیرد آن است که آیا سرور بدون حالت<sup>۳۷۴</sup> است یا خیر. سرور بدون حالت ، اطلاعات مربوط به وضعیت مشتریان خود را نگه نداشته و می توانند بدون اطلاع مشتریان تغییر وضعیت دهد (Birman، ۲۰۰۵). بعنوان مثال، سرورهای وب بدون حالت بوده و صرفاً به درخواستهای HTTP ورودی پاسخ می دهند که ممکن است یا مربوط به آپلود کردن فایل به سرور باشند و یا (در اغلب موارد) جهت واکشی فایل. پس از پردازش درخواست، سرور وب کاملاً مشتری را فراموش می کند. همچنین می توان مجموعه فایلهایی را که یک سرور (احتمالاً با همکاری با یک سرور فایل) مدیریت می کند ، بدون نیاز به مطلع ساختن مشتریان تغییر داد.

توجه داشته باشید که در بسیاری از طرح های بدون حالت، سرور عملاً اطلاعاتی از مشتریان خود را نگه می دارد ، اما واقعیت مهم این است که مفقود شدن این اطلاعات باعث وقفه در سرویس ارائه شده از طرف سرور نخواهد شد. بعنوان مثال، یک سرور وب غالباً تمامی درخواستهای مشتریان را ثبت می کند.

<sup>371</sup> interrupt

<sup>372</sup> out-of-band

<sup>373</sup> urgent

<sup>374</sup> Stateless

این اطلاعات مثلاً در هنگام تصمیم گیری در مورد لزوم یا عدم لزوم کپی برداری برخی اسناد و محل کپی شدن آنها ممکن است مورد استفاده قرار گیرد. ناگفته پیداست که در صورت مفقود شدن ثبتی ها، احتمالاً تنها مشکل آن است که کارآیی کمتر از حد بهینه خواهد شد.

حالت خاصی از طراحی بدون حالت هنگامی است که سرور دارای **حالت نرم**<sup>۳۷۵</sup> باشد. در این حالت، سرور متعهد می شود که به نمایندگی از جانب مشتری و فقط برای مدت زمانی محدود، وضعیت مشتری را حفظ کند. پس از منقضی شدن این زمان، سرور به رفتار پیش فرض خود بازگشته و تمامی اطلاعاتی را که از مشتری مورد نظر در اختیار دارد، دور خواهد ریخت. بعنوان یک نمونه از این نوع حالت، می توان سروری را ذکر کرد که تهعد نموده تا مشتری مفروض را برای مدت زمانی کوتاهی از به هنگام سازی ها آگاه نگه دارد. پس از گذشت این زمان، مشتری باید درمورد به هنگام سازی ها از سرور پرسش نماید. رویکردهای هویت نرم گرچه از طراحی پروتکل در شبکه های کامپیوتری اقتباس شده اند، اما می توانند به همان گونه در طراحی سرور هم کاربرد داشته باشند (Clark, ۱۹۹۸؛ ۲۰۰۴؛ Lui و گروه همکاران).

بر عکس، **سرور دارای حالت**<sup>۳۷۶</sup> غالباً اطلاعات ماندگار از مشتریان خود را نگه می دارد. به این معنی که اطلاعات باید صریحاً به توسط سرور حذف شوند. بعنوان یک نمونه می توان سرور فایلی را ذکر کرد که به مشتری اجازه می دهد تا حتی برای انجام عملیات به هنگام سازی، کپی محلی از فایل را در اختیار داشته باشد. چنین سروری جدولی شامل درایه های (مشتری، فایل) را نگه خواهد داشت. این جدول به سرور امکان می دهد تا گزارش لحظه ای از این که کدام مشتری اجازه به هنگام سازی کدام فایل را دارد در اختیار داشته باشد، و احتمالاً همان مشتری آخرین نسخه از آن فایل را هم در اختیار دارد.

این روش می تواند باعث افزایش کارآیی عملیات خواندن و نوشتمن از دیدگاه مشتری شود. افزایش کارآیی در قیاس با سرورهای بدون حالت یکی از مزایای مهم طرح های دارای حالت تلقی می شود. در عین حال، مثال گفته شده بیانگر نقطه ضعف عمده سرورهای دارای حالت هم می باشد. در صورت شکست سرور بایستی جدول درایه های (مشتری، فایل) خود را بازسازی کند، والا نمی تواند تضمین کند که آخرین به هنگام سازی ها را روی فایل پردازش کرده باشد. در کل، سرور دارای حالت باید تمامی وضعیت خود را بازسازی کرده و به حالت پیش از خرابی بازگردد. همانطور که در فصل ۸ هم خواهیم گفت، ایجاد امکان بازسازی پیچیدگی های زیادی دارد. در طرح بدون حالت، برای بازسازی سرور از کارافتاده نیازمند انجام هیچ اقدام خاصی نیستیم. بلکه فقط مجدداً راه اندازی شده و سرور منتظر ورود درخواستهای مشتری باقی خواهد ماند.

<sup>375</sup> soft state

<sup>376</sup> Stateful

Ling (۲۰۰۴) و گروه همکاران استدلال می کنند که باید بین **حالت جلسه**<sup>۳۷۷</sup> (موقعت) و **حالت دائم** تمایز قائل شویم. مثال گفته شده نمونه ای از حالت جلسه است و در رابطه با مجموعه عملیات صورت گرفته توسط یک کاربر است که باید برای مدت زمانی معین، اما نه نامشخص، حفظ شود. در نتیجه، **حالت جلسه غالباً** در معماری های مشتری- سرور سه طبقه ای<sup>۳۷۸</sup> استفاده می شود که در آنها، سرور برنامه کاربردی بایستی پیش از پاسخ به مشتری درخواست کننده، از طریق مجموعه ای از پرس و جوها به سرور پایگاه داده مناسبی دسترسی پیدا کند. نکته این است که در صورت از دست رفتن **حالت جلسه**، مشروط براینکه مشتری بتواند درخواست اصلی را مجدداً صادر نماید، اشکالی بوجود نخواهد آمد.

این مشاهده امکان ذخیره سازی ساده تر ولی با قابلیت اطمینان کمتر **حالت را مطرح می کند**.

آنچه در وضعیت دائم باقی می ماند غالباً اطلاعاتی است که در پایگاه های داده نگهداری می شوند، از قبیل اطلاعات خریداران، کلیدهای مربوط به نرم افزار خریداری شده و غیره. با این وجود، در اغلب سیستم های توزیعی، حفظ **حالت جلسه** بیانگر یک طرح دارای **حالت** است که در صورت وقوع خرابی ها **حتماً** نیازمند انجام اقدامات خاص و همچنین ارائه فرضیات آشکار در مورد ماندگاری **حالت ذخیره** شده در سرور می باشد. در مبحث مربوط به تحمل خرابی راجع به این مسائل بحث خواهیم کرد.

در هنگام طراحی سرور، انتخاب هریک از گزینه های طراحی بدون **حالت** و دارای **حالت** نبایستی بر خدمات ارائه شده توسط سرور تأثیر بگذارد. بعنوان مثال، در صورتی که باید فایل ها پیش از خوانده شدن یا نوشته شدن باز شوند، پس یک سرور بدون **حالت** باید به گونه ای این رفتار را تقلید کند. یک راه حل معمول که با جزئیات بیشتر در فصل ۱۱ بحث خواهد شد این است که ابتدا سرور با بازکردن فایل ارجاع داده شده در برابر درخواست نوشتن یا خواندن واکنش داده و سپس واقعاً عملیات خواندن و نوشتن را انجام دهد و فایل را فوراً دوباره ببندد.

در بقیه موارد، ممکن است سرور بخواهد گزارش رفتار مشتریان خود را نگهداری کند تا بتواند به نحو مؤثرتری به درخواست های آنان پاسخ دهد. بعنوان مثال، سرورهای وب برخی اوقات می توانند مشتری را مستقیماً به صفحات مورد علاقه او هدایت کنند. این روش فقط در صورتی قابل اجرا خواهد بود که سرور سابقه ای از آن مشتری در اختیار داشته باشد. زمانی که سرور قادر به حفظ وضعیت نباشد، یک راه حل معمول این است که مشتری اطلاعات اضافی درمورد دسترسی های قبلی خود ارسال کند. در مورد وب، این اطلاعات غالباً به صورتی شفاف توسط جستجوگر مشتری در چیزی به نام **کوکی**<sup>۳۷۹</sup> ذخیره خواهد شد؛ کوکی در واقع قطعه کوچکی از داده های حاوی اطلاعات ویژه مشتری است که سرور به آنها علاقه مند می باشد. این کوکی ها هرگز توسط جستجوگر اجرا نشده و صرفاً ذخیره می شوند.

<sup>377</sup> session

<sup>378</sup> Three-tiered

<sup>379</sup> cookie

وقتی مشتری برای اولین بار به سرور دسترسی پیدا می کند، سرور به همراه صفحات وب درخواستی یک کوکی را هم به جستجوگر باز می گرداند و او هم کوکی را برای استفاده های بعدی در جایی ذخیره می کند . از آن به بعد، هر زمانیکه مشتری به سرور دسترسی پیدا می کند، کوکی مربوط به آن سرور به همراه درخواست برای او ارسال می شود. این روش در مجموع کارآیی مناسبی دارد و این واقعیت که کلوچه ها برای محافظت توسط جستجوگر عودت داده می شوند بطور کامل از کاربران مخفی نگه داشته می شود. البته این کار در تناظر با حفظ حریم خصوصی است.

### ۳-۴-۲- خوشه های سرور<sup>۳۸۰</sup>

در فصل ۱ مختصرأ راجع به محاسبات خوشه ای بعنوان یکی از چندین نوع نمای سیستم های توزیعی بحث کردیم. در اینجا نگاه دقیقتری خواهیم داشت به سازمان خوشه های سرور و همچنین مهمترین مشکلات مسائل طراحی این خوشه ها.

#### سازمان عمومی

به بیان ساده، خوشه سرور در واقع مجموعه ای از ماشینهایی است که از طریق یک شبکه به هم متصل شده و هر ماشین یک یا چند سرور را راه اندازی می کند. خوشه های سروری که تا به اینجا مورد بررسی قرار گرفت از جمله خوشه هایی هستند که ماشینهای آنها از طریق یک شبکه محلی به هم متصل شده و غالباً پهنای باند زیاد ، ولی تأخیر ناچیزی را ایجاد می کنند.

در اغلب موارد و مطابق شکل ۱۲-۳، خوشه سرور از نظر منطقی بصورت سه طبقه سازماندهی می شود. طبقه اول متشکل از سوئیچ (منطقی) است که درخواستهای مشتری از طریق آن مسیریابی می شوند. این سوئیچها ممکن است بسیار متفاوت باشند. بعنوان مثال، همانطور که در ادامه هم خواهیم گفت، سوئیچهای لایه انتقال درخواستهای اتصال TCP ورودی را پذیرفته و آنها را به یکی از سرورهای داخل خوشه عبور می دهند. یک نمونه کاملاً متفاوت دیگر سرور وبی است که گرچه درخواستهای HTTP ورودی را می پذیرد ، بخشی از درخواست ها را جهت پردازش بیشتر به سرورهای برنامه کاربردی ارسال می کند تا بعداً بتواند نتایج را جمع آوری و پاسخی را به HTTP بازگرداند.

شکل ۱۲-۳- سازمان کلی یک خوشه سرور سه طبقه ای (داخل شکل: طبقه اول، سوئیچ منطقی (احتمالاً متعدد)، درخواست های مشتری، درخواست اعزام شده، طبقه دوم، سرورهای برنامه کاربردی / محاسبه، طبقه سوم، سیستم پایگاه داده ای / فایل توزیع شده).

مشابه تمامی معماری مشتری - سرور چندطبقه ای ، بسیاری از خوشه های سرور حاوی سرورهای اختصاص یافته جهت پردازش برنامه کاربردی هستند. در محاسبه خوشه ای، این سرورها نوعاً روی سخت افزار خاص با کارآیی بالا برای تحويل توان محاسباتی بالا اجرا می شوند. اما در خوشه های سرور

<sup>380</sup> server clusters

سازمانی<sup>۳۸۱</sup> ، از آنجایی که مشکل کار دسترسی به منبع است نه کسب توان محاسباتی ، ممکن است لازم باشد که برنامه های کاربردی فقط روی ماشینهای نسبتاً پایین اجرا شوند.

طبقه سوم متشکل از سرورهای پردازش داده و مخصوصاً، سرورهای فایل و پایگاه داده ای است. این سرورها هم بسته به کاربرد خوش سرور، ممکن است روی ماشینهای خاص منظوره اجرا شده و برای دسترسی پرسرعت به دیسک پیکربندی شوند و از حافظه های مخفی بزرگ برای داده های طرف سرور استفاده کنند.

البته تمام خوش های سرور از این دسته بندی کاملاً مشخص پیروی نمی کنند. در بسیاری از موارد، هر ماشین مجهز به منبع ذخیره محلی خود بوده و غالباً، برنامه کاربردی و پردازش داده را در یک سرور تجمیع کرده و در نتیجه معماری دوطبقه ای ایجاد خواهد شد. عنوان مثال، در مبحث جریان رسانه<sup>۳۸۲</sup> با استفاده از خوش سرور، معمولاً رسم براین است که از معماری سیستم دوطبقه ای استفاده شده و هر ماشین، عنوان یک سرور رسانه اختصاصی شده عمل می کند Steinmetz and Nahrstedt (۲۰۰۴).

در صورتیکه یک خوش سرور چندین سرویس را ارائه کند، ممکن است ماشینهای مختلف سرورهای برنامه کاربردی مختلفی را اجرا کنند. در نتیجه، سوییچ قادر خواهد بود تا بین سرویسها تمایز قائل شود، و الا قادر به ارسال درخواستها به ماشینهای مربوطه نخواهد بود. در نتیجه، بسیاری از ماشین های طبقه دوم فقط یک برنامه کاربردی را اجرا می کنند. این محدودیت علاوه بر دلالت بر وابستگی به سخت افزار و نرم افزار موجود ، ناشی از این واقعیت هم می باشد که برنامه های کاربردی مختلف غالباً بوسیله مدیران اجرایی مختلف مدیریت می شوند. این مدیران هم تمایلی به دخالت در کار ماشین های یکدیگر ندارند.

در نتیجه مشاهده می شود که برخی ماشینها موقتاً بیکار هستند ، در حالی که درخواست های بسیار به برخی دیگر باعث اضافه بار آنها می شود. در اینصورت بهتر است که سرویسها را موقتاً به ماشینهای بیکار مهاجرت دهیم. راه حل پیشنهادی در Awadallah and Rosenblum (۲۰۰۴) ، استفاده از ماشینهای مجازی جهت مهاجرت نسبتاً آسان کد به ماشینهای واقعی است. در بخش های بعدی همین فصل به مسئله مهاجرت کد باز خواهیم گشت.

حال باید طبقه اول را که حاوی سوییچ است، به دقت بررسی کنیم. یکی از اهداف مهم در طراحی خوش های سرور، مخفی سازی واقعیت وجود چندین سرور است. به بیان دیگر، برنامه های کاربردی که روی ماشینهای مختلف اجرا می شوند، قاعدها نیازی به اطلاعات راجع به سازمان داخلی خوش ندارند. این شفافیت دسترسی بوسیله یک نقطه دسترسی ارائه می شود که آنهم متقابلاً از طریق نوعی سوییچ سخت افزار مانند یک ماشین اختصاصی پیاده سازی می شود.

<sup>381</sup> enterprise

<sup>382</sup> Streaming Media

سوئیچ در واقع نقطه ورودی برای خوش سرور بوده و تنها یک آدرس شبکه ای دارد. جهت ایجاد قابلیت های مقیاس پذیری و دسترسی پذیری، خوش سرور ممکن است چندین نقطه دسترسی داشته باشد که هریک بوسیله یک ماشین اختصاصی مشخص می شود. ما در اینجا فقط مورد یک نقطه دسترسی را در نظر می گیریم.

یکی از روش های استاندارد برای دسترسی به خوش سرور ، ایجاد اتصال TCP است تا پس از آن درخواست های سطح برنامه کاربردی از طریق آن و عنوان بخشی از جلسه ارسال شوند. پایان جلسه با شکستن اتصال اعلان می شود. در مورد **سوئیچ های لایه انتقال**، سوئیچ درخواستهای اتصالات TCP ورودی را پذیرفته و این اتصال را به یکی از سرورها تقدیم<sup>۳۸۳</sup> می کند (Hunt، ۱۹۹۸ و گروه همکاران؛ Pai، ۱۹۹۸ و گروه همکاران). اصل کلی چیزی که معمولاً به نام **تقدیم کردن TCP** خوانده می شود، در شکل ۳-۱۳ ارائه شده است.

**شکل ۳-۱۳- اصل کلی تقدیم TCP** (داخل شکل: یک اتصال منطبقاً جداگانه TCP مشتری، درخواست، پاسخ، سرور، سوئیچ، (تقدیم) درخواست، سرور).

پس از آنکه سوئیچ درخواست اتصال TCP را دریافت می کند، بهترین سرور را برای تقدیم آن درخواست انتخاب و بسته درخواست را به آن سرور ارسال می کند. سرور هم متقابلاً تصدیقی را به مشتری درخواست کننده می دهد، و همچنین آدرس IP سوئیچ را در حوزه منبع سرآیند بسته IP حامل قطعه TCP درج می نماید. توجه داشته باشید که این ترفند برای ادامه اجرای پروتکل TCP توسط مشتری ضروری است؛ چون مشتری در انتظار پاسخ از جانب سوئیچ است و نه از جانب سرور دیگری که تا به حال حتی نامی از آن برده نشده. مشخص است که پیاده سازی تقدیم TCP نیازمند انجام برخی اصلاحات در سطح سیستم عامل است.

حتماً تا به اینجا متوجه شده اید که سوئیچ می تواند نقش مهمی در توزیع بار در بین سرورهای مختلف داشته باشد. با تصمیم گیری در مورد محل ارسال درخواست، سوئیچ در مورد سرور مسئول انجام پردازش بیشتر روی درخواست هم تصمیم گیری می کند. ساده ترین سیاست برای تعادل بار آن است که سوئیچ از روش نوبت گردشی بهره ببرد: یعنی هربار سرور بعدی را از لیست خود انتخاب و درخواست را برای آن ارسال کند.

از ضوابط پیشرفته تری هم می توان برای انتخاب سرور استفاده نمود. عنوان مثال، فرض کنید که یک خوش سرور سرویس‌های متعددی را ارائه می دهد. چنانچه سوئیچ بتواند در صورت ورود درخواست باز هم بین این سرویس‌ها تمایز قائل شود، بنابراین قادر خواهد بود که تصمیمات آگاهانه ای در مورد محل ارسال درخواست اتخاذ کند. انتخاب سرور می تواند همچنان در لایه انتقال انجام شود، البته مشروط به اینکه سرویس ها بوسیله شماره درگاه از هم تفکیک شده باشند. گام بعدی آن است که اجازه دهیم

<sup>383</sup> Hand-off

سوئیچ ماموریت درخواست ورودی را مشخص کند. از این روش فقط در صورتی می‌توان استفاده نمود که بداند ماموریت چگونه به نظر می‌رسد. بعنوان مثال، درمورد سرورهای وب، سوئیچ نهایتاً در انتظار یک درخواست HTTP است تا براساس آن درمورد واحد پردازش کننده آن تصمیم بگیرد. در فصل ۱۲ در مبحث مربوط به سیستم‌های براساس وب مجدداً به موضوع **توزیع درخواست محتوا آگاه**<sup>۳۸۴</sup> بازخواهیم گشت.

### ۳۸۵ سرورهای توزیعی

اغلب خوش‌های سروری که تا به اینجا مورد بحث قرار گرفت، بیشتر به صورت آماری پیکر بندی شده بودند. در این خوش‌ها، معمولاً<sup>۳۸۵</sup> یک ماشین مدیریت اجرایی جداگانه وجود دارد که گزارش سرورهای موجود را نگهداری کرده و به نحو مقتضی، این اطلاعات را به ماشین‌های دیگری از قبیل سوئیچ می‌دهد.

همانطور که قبل‌اهم گفتیم، اغلب خوش‌های سرور فقط یک نقطه دسترسی ارائه می‌کنند. در صورت خرابی این نقطه، خوش‌هم غیرقابل دسترسی خواهد شد. برای حذف این مشکل احتمالی، می‌توان چندین نقطه دسترسی ایجاد نموده و آدرس‌های آنها را در اختیار عموم قرار داد. بعنوان مثال، سیستم نام دامنه<sup>۳۸۶</sup>(DNS) می‌تواند آدرس‌های متعددی را بازگرداند که همگی متعلق به یک نام میزبان واحد باشند. در این روش هم مشتریان باید در صورت خراب شدن یکی از آدرسها، مجدداً تلاش‌های متعددی بنمایند. بعلاوه، راه حلی برای مشکل نیاز به نقاط دسترسی ثابت ارائه نمی‌دهد.

داشتن ثبات، مانند داشتن نقطه دسترسی با عمر طولانی، یکی از ویژگی‌های مطلوب از نقطه نظر مشتری و سرور است. از طرف دیگر، بهتر است در پیکر بندی خوش‌سرور (شامل سوئیچ) از انعطاف پذیری بالایی برخوردار باشیم. این مشاهده منجر به طراحی سرور توزیعی شده که عملآ مجموعه‌ای از ماشین‌ها است که تعداد آنها به صورت پویا تغییر می‌کند و احتمالاً دارای نقاط دسترسی متغیری است؛ اما برای ناظران بیرونی فقط یک ماشین منفرد و قدرتمند می‌باشد. طراحی این دست سرورهای توزیعی در کتاب (۲۰۰۵) Szymaniak و گروه همکاران ارائه شده است. ذیلاً مختصراً در مورد آن خواهیم گفت.

ایده اصلی در سرورهای توزیعی آن است که مشتریان از مزایای وجودی یک سرور مقاوم، با کارآیی بالا و پایدار برخودار شوند. این ویژگی‌ها را غالباً به وسیله کامپیوترهای بزرگ به دست می‌آورند که برخی، میانگین زمانی بین دو خرابی آنها را حتی تا بیش از ۴۰ سال هم ذکر کرده‌اند. اما، با دسته بندی شفاف ماشین‌های ساده‌تر در یک خوش‌ه و تکیه نکردن بر موجود بودن فقط یک ماشین، ممکن است بتوان به درجه پایداری بهتری در مقایسه با هریک از مؤلفه‌ها دست یافت. بعنوان مثال، چنین

<sup>384</sup> content-aware

<sup>385</sup> distributed servers

<sup>386</sup> domain name system

خوشه ای را می توان به صورت پویا با استفاده از ماشین های کاربران پیکربندی نمود، همچنان که در مورد سیستم های توزیعی همکاری کننده<sup>۳۸۷</sup> عمل می شود.

حال ببینیم چگونه می توان در این دست سیستم ها یک نقطه دسترسی ثابت<sup>۳۸۸</sup> ایجاد کرد . ایده اصلی استفاده از سرویس های شبکه سازی موجود، خصوصاً پشتیبانی جابجایی برای نسخه ۶ IP<sup>۳۸۹</sup> (MIPv6) است. در MIPv6، فرض می شود که یک گره متحرک دارای یک شبکه خانگی<sup>۳۹۰</sup> است که معمولاً در آن سکونت داشته و به ازای آن، آدرس ثابتی به نام آدرس خانه<sup>۳۹۱</sup> (HoA) دارد. به این شبکه خانگی، مسیریاب خاصی به نام نماینده خانه<sup>۳۹۲</sup> متصل است که مراقب جریان ترافیک به سمت گره متحرک، در صورت دور بودن از آن است. وقتی یک گره متحرک به یک شبکه خارجی متصل می شود، آدرس مراقبتی<sup>۳۹۳</sup> (CoA) موقتی دریافت می کند که از آن طریق به گره دسترسی می شود. این آدرس مراقبتی به نماینده خانه گره گزارش شده و آنهم مراقب است که از آن پس تمام ترافیک به گره متحرک پیش برده شود. توجه داشته باشید که برنامه های کاربردی که با گره متحرک ارتباط برقرار می کنند، فقط آدرس شبکه خانگی گره را درک کرده و هرگز آدرس مراقبتی را مشاهده نخواهند کرد.

از این اصل کلی می توان جهت دادن آدرسی ثابت به سرورهای توزیعی استفاده نمود. در این حالت، ابتدا یک آدرس تماس<sup>۳۹۴</sup> منحصر به فرد و جداگانه به خوش سرور نسبت داده می شود. این آدرس تماس، آدرس عمری سرور است که در تمام ارتباطات آن با دنیای خارج مورد استفاده قرار خواهد گرفت. در هر زمان، یکی از گره های داخل سرور توزیعی بعنوان نقطه دسترسی این آدرس تماس عمل خواهد کرد؛ اما این نقش قابل واگذاری به گره های دیگر هم می باشد . در واقع داستان به این صورت است که، نقطه دسترسی مفروض آدرس خود را بعنوان آدرس مراقبتی در نماینده خانه سرور توزیعی ثبت می کند. در این مقطع ، تمام ترافیک به نقطه دسترسی هدایت خواهد شد و نقطه هم از آن پس، مراقب توزیع در خواستها در بین گره هایی است که در همان زمان در خوش هستند. در صورت خرابی نقطه دسترسی، یک مکانیزم کنترل خرابی ساده وارد صحنه شده و به این ترتیب ، یک نقطه دسترسی دیگر آدرس مراقبتی جدید را گزارش خواهد داد.

در همین پیکربندی ساده ممکن است نماینده خانه و نقطه دسترسی تبدیل به نقطه شکست یکتا شوند، چون تمام ترافیک از طریق ایندو ماشین جاری خواهد شد. می توان بکمک یکی از ویژگی های

<sup>387</sup> collaborative

<sup>388</sup> stable access point

<sup>389</sup> Mobile IP version 6

<sup>390</sup> home network

<sup>391</sup> home address

<sup>392</sup> home agent

<sup>393</sup> care-of address

<sup>394</sup> contact address

MIPv6 به نام بهینه سازی مسیر<sup>۳۹۵</sup> از بروز این مشکل جلوگیری کرد. بهینه سازی مسیر به اینصورت عمل خواهد کرد: هنگامیکه یک گره متوجه با آدرس خانگی  $HA$  آدرس مراقبتی فعلی خود ، مثلاً  $CA$  ، را گزارش می دهد، نمایندگی خانه می تواند  $CA$  را برای یک مشتری ارسال نماید. مشتری هم  $CA$  جفت ( $HA, CA$ ) را به صورت محلی ذخیره خواهد کرد . از این لحظه به بعد، ارتباطات مستقیماً برای  $CA$  پیش برد خواهد شد. هرچند برنامه کاربردی طرف مشتری هنوز هم می تواند از آدرس خانگی استفاده کند، نرم افزار پشتیبان زیربنایی MIPv6 این آدرس را به  $CA$  ترجمه کرده و بجای آن مورد استفاده قرار خواهد داد.

شكل ۳-۱۴- بهینه سازی مسیر در یک سرور توزیعی( داخل شکل: مشتری ۱، باور می کند که سرور آدرس  $HA$  را در اختیار دارد، باور می کند که سرور به  $X$  متصل است، باور می کند که  $X$  در محل  $CA1$  مستقر است، مشتری ۲، باور می کند که سرور آدرس  $HA$  را در اختیار دارد، باور می کند که سرور به  $X$  متصل است، باور می کند که  $X$  در محل  $CA2$  مستقر است، سرور توزیعی  $X$ ، سرور ۱، می داند که مشتری ۱ باور می کند که  $X$  است، نقطه دسترسی با آدرس  $CA1$ ، سرور ۲، نقطه دسترسی با آدرس  $CA2$ ، می داند که مشتری ۲ باور می کند که  $X$  است، اینترنت).

مطابق شکل ۳-۱۴ ، از بهینه سازی مسیر می توان برای قبولاندن این مطلب به مشتریان استفاده کرد که با یک سرور واحد تماس دارند، در حالی که هر مشتری در ارتباط با گره مجزایی از سرور توزیعی است. هنگامیکه یکی از نقاط دسترسی داخل سرور توزیعی درخواستی را از مشتری  $C_1$  به مثلاً گره  $S_1$  ( با آدرس  $CA_1$  ) پیش می برد، آنقدر اطلاعات به  $S_1$  عبور می دهد که برای آغاز روال بهینه سازی مسیر کفایت کند. بر اساس این اطلاعات، مشتری نهایتاً باور می کند که آدرس مراقبتی،  $CA_1$  است. به این ترتیب،  $C_1$  قادر به ذخیره سازی جفت ( $HA, CA_1$ ) می شود. در جریان این روال، نقطه دسترسی ( و همچنین نماینده خانه ) بعنوان تونل هدایت عمده ترافیک بین  $C_1$  و  $S_1$  عمل می کند. این کار از این که نماینده خانه باور کند تغییر آدرس مراقبتی داشته است جلوگیری می کند و همچنان به ارتباط خود با نقطه دسترسی ادامه دهد.

البته، در حین انجام روال بهینه سازی مسیر، ممکن است درخواست هایی از جانب مشتریان دیگر وارد شود. این درخواست ها در نقطه دسترسی در وضعیت معلق باقی خواهند ماند تا این که بتوانند پیش بردشوند. پس ممکن است درخواست یکی دیگر از مشتریان به نام  $C_2$  به سوی گره عضو  $S_2$  (با آدرس  $CA_2$  ) پیش برد شده و  $S_2$  اجازه دهد مشتری  $C_2$  جفت ( $HA, CA_2$ ) را ذخیره کند . در نتیجه، مشتریان مختلف مستقیماً با اعضای مختلف سرور توزیعی ارتباط برقرار می کنند، در حالی که

<sup>395</sup> route optimization

همچنان هر برنامه کاربردی مشتری این تصور را دارد که سرور آدرس  $HA$  را دارد. نماینده خانگی همچنان به ارتباط خود با نقطه دسترسی در حال گفتگو با آدرس تماس ادامه خواهد داد.

### ۳-۴-۳- مدیریت خوشه های سرور

یک سرور خوشه ای باید برای دنیای خارج مانند یک کامپیوتر منفرد بنظر برسد- که در عمل غالباً هم همینطور است. با این وجود، وقتی کار به مدیریت خوشه ها می رسد، شرایط به شدت تغییر می کند. همانطور که در ادامه خواهیم گفت، تلاش‌های زیادی برای تسهیل مدیریت خوشه های سرور انجام گرفته است.

#### روشهای متعارف

تا به حال ، متعارف ترین روش در مدیریت خوشه های سرور، بسط کارکردهای سنتی مدیریت تک کامپیوترها به خوشه بوده است. در ساده ترین شکل ، مدیر اجرایی در گره یکی از مشتریان دور ثبت نام<sup>۳۹۶</sup> کرده و فرامین مدیریت محلی مربوط به نظارت، نصب و تغییر مؤلفه ها را اجرا می کند. در حالت کمی پیشرفته تر ، ثبت نام در یک گره مفروض مخفی می شود و به جای آن یک واسط در ماشین مدیر اجرایی ارائه می شود که امکان جمع آوری اطلاعات از یک یا چند سرور ، ارتقاء مؤلفه ها ، حذف یا اضافه گره ها و غیره را فراهم می آورد. مزیت عمدۀ این روش ، تسهیل عملیات جمیع است که بر روی گروهی از سرورها انجام می شود. این نوع مدیریت خوشه های سرور کاربرد عملی فراوانی پیدا کرده و از نمونه های آن می توان به نرم افزار مدیریتی از قبیل مدیریت سیستم های خوشه ای<sup>۳۹۷</sup> اشاره کرد(Hochstetler and Beringer ۲۰۰۴).

با این وجود، چنانچه تعداد خوشه ها به بیش از چند ده گره برسد، این نوع مدیریت دیگر کارساز نخواهد بود. بسیاری از مراکز داده ای مدیریت هزاران سروری را بر عهده دارند که گرچه به شکل چندین خوشه سازماندهی شده اند، اما همگی با همکاری عمل می کنند. انجام اینکار بوسیله سرورهای مدیریت اجرایی مرکز غیرممکن است. بعلاوه، به راحتی می توان مشاهده کرد که خوشه های بسیار بزرگ نیازمند مدیریت تعمیر دائمی (شامل ارتقاء ها) هستند. به بیان ساده تر، چنانچه  $p$  احتمال خراب بودن یک سرور بوده و با فرض مستقل بودن خرابی ها، آنگاه برای آنکه خوشه ای با  $N$  سرور بدون خرابی حتی یک سرور عمل کند،  $(I-p)^N$  خواهد بود. چنانچه  $1 - (I-p)^N = 0.000$  و احتمال عملکرد صحیح تمام سرورها فقط ۳۶ درصد خواهد بود.

در نتیجه، پشتیبانی از خوشه های سرور بسیار بزرگ تقریباً همیشه موردنی است. در این رابطه ، از قواعد تجربی زیادی می توان استفاده کرد (Brewer ۲۰۰۱)، اما روش سیستماتیکی برای کار بر روی موضوع مدیریت سیستم های بسیار بزرگ وجود ندارد. مدیریت خوشه ها هنوز در مراحل اولیه است، اما

<sup>396</sup> Login

<sup>397</sup> Cluster System Management

می توان امیدوار بود که پس از کسب تجربیات کافی در مورد راه حلهای خود مدیریتی، که در فصل قبل مورد بررسی قرار گرفت، نهایتاً در این موضوع راه خود را بیابد.

### مثال: **PlanetLab**

حال بباید نگاه دقیقتری به یک سرور خوشه ای تقریباً غیر معمول داشته باشیم . **PlanetLab** یک سیستم توزیعی براساس همکاری است که در آن ، هر یک از سازمانها یک یا چند کامپیوتر را به جمع صدها گره موجود هدیه می دهند . این کامپیوترها رویهمرفته خوش سرور تک طبقه ای ایجاد می کنند که در آن تمامی امور دسترسی، پردازش و ذخیره سازی می توانند ببروی هریک از گره ها انجام شوند. مدیریت **PlanetLab** در اغلب موارد لزوماً توزیعی است. پیش از معرفی اصول اساسی، ابتدا ویژگی های اصلی این سیستم توزیعی را تشریح می کنیم ( Peterson و گروه همکاران).

در **PlanetLab**، هر سازمان یک یا چند گره هدیه می کند؛ گره می تواند صرفاً یک کامپیوتر باشد، اگرچه این امکان هست که خوشه ای متشکل از چندین ماشین هم باشد. نحوه سازماندهی گره ها در شکل ۳-۱۵ ارائه شده است . در این رابطه دو مؤلفه مهم وجود دارد( ۲۰۰۴ Bavier و گروه همکاران). اول این که، ناظر ماشین مجازی (VMM)<sup>۳۹۸</sup> یک سیستم عامل بهبود یافته Linux است. بهبود یافته‌گی عمدتاً شامل تنظیماتی برای پشتیبانی از مؤلفه دوم یعنی V سرور ها است. V سرورهای Linux را می توان عنوان محیط مجازی تلقی کرد که گروهی از فرآیندها در آن در حال اجرا هستند. فرآیندهای مربوط به V سرورهای مختلف کاملاً مستقل بوده و نمی توانند مستقیماً در تمام منابع از جمله فایلها، حافظه اصلی و اتصالات شبکه سهیم شوند، چیزی که در بین فرآیندهای در حال اجرا بر روی یک سیستم عامل وجود دارد. بلکه در عوض، V سرور محیطی متشکل از مجموعه از بسته های نرم افزاری خودی، برنامه ها و امکانات شبکه سازی است. به عنوان مثال، یک V سرور می تواند محیطی فراهم آورد که در آن یکی از فرآیندها درمی یابد که می تواند از ترکیب Python 1.5.2 با وب سرور قدیمی تری از نوع Apache، مثلاً httpd 1.3.3 استفاده کند. در حالیکه سرور دیگری ممکن است از آخرین نسخه های Python و httpd پشتیبانی کند. البته اطلاق نام " سرور " به یک V سرور تاحدودی اشتباه است ، چون عملاً فقط جدایی گروه های فرآیندها از یکدیگر را باعث می شود. کمی بعدتر مجدداً به مبحث V سرورها بازخواهیم گشت.

شکل ۳-۱۵ - سازمان اصلی یک گره **PlanetLab** ( داخل شکل: ماشینهای مجازی تخصیص یافته به کاربر، ماشینهای مجازی مدیریت ممتاز، فرآیند، V سرور، سیستم عامل بهبود یافته Linux. سخت افزار).

Linux VMM تضمین می کند که V سرورها از هم جدا باشند: فرآیندهای داخل V سرور های مختلف به صورتی همرونده و مستقل از یکدیگر اجرا می شوند، و هریک از آنها فقط از بسته های نرم

<sup>398</sup> virtual machine monitor

افزاری و برنامه های موجود در محیط خود استفاده می کند. مرزهای جداگننده بین فرآیندها در سرورهای مختلف جدی هستند. عنوان مثال، ممکن است دو فرآیند در ۷ سرورهای مختلف ID کاربر یکسانی داشته باشند، اما این به هیچ وجه معنای داشتن کاربر واحد نیست. قائل شدن این تفکیک به میزان چشمگیری باعث تسهیل پشتیبانی کاربران از سازمانهای مختلفی خواهد شد که می خواهند از PlanetLab ، مثلاً ، عنوان بستری برای تجربه برنامه های کاربردی و سیستم های توزیعی کاملاً متفاوت استفاده کنند.

جهت پشتیبانی از چنین تجربیاتی ایده برش<sup>۳۹۹</sup> را مطرح می کند که در واقع، مجموعه ای از ۷ سرورهایی است که هریک روی گره متفاوتی اجرا می شوند. بنابراین، هر برش را می توان خوش سرور مجازی تصور کرد که بوسیله مجموعه ای از ماشینهای مجازی پیاده می شود. ماشینهای مجازی در PlanetLab بر روی سیستم عامل Linux اجرا می شوند که هسته اصلی سیستم عامل با پیمانه های توسعه یافته است.

به دلیل وجود مسائل متعدد، که از مهمترین آنها می توان به ۳ مورد زیر اشاره کرد، مدیریت PlanetLab تبدیل به مسأله ویژه ای شده است:

۱. گره ها به سازمانهای مختلفی تعلق دارند. هر یک از این سازمانها بایستی قادر به تعیین فرد مجاز برای اجرای برنامه های کاربردی بر روی گره های خود بوده ، و همچنین کاربرد منابع را به نحو مناسبی محدود کنند.
  ۲. گرچه ابزارهای نظارتی مختلفی وجود دارد، اما هریک ترکیب خاصی از سخت افزار و نرم افزار را فرض می کنند. بعلاوه، هر یک به صورت سفارشی برای یک سازمان قابل استفاده است.
  ۳. برنامه های مربوط به برشهای متفاوت که روی یک گره واحد اجرا می شوند، نباید در کار یکدیگر مداخله کنند. این مورد مشابه استقلال فرآیند در سیستم عامل است.
- در ادامه هریک از مسائل فوق را مفصلًا مورد بررسی قرار خواهیم داد.

در مدیریت منابع PlanetLab، نقش کلیدی بر عهده مدیر گره است. تمامی گره ها از وجود چنین مدیری برخوردار هستند. مدیر گره بوسیله یک ۷ سرور جداگانه اجرا شده و تنها وظیفه آن ایجاد سرورهای دیگر روی گره تحت مدیریت خود و کنترل تخصیص منابع است. مدیر گره هیچگونه تصمیمی در مورد سیاست ها نمی گیرد و فقط مکانیزمی است برای ایجاد اجزاء اساسی لازم جهت اجرای برنامه روی یک گره مفروض.

پی گیری منابع بوسیله یکی از ویژگی های منبع ، یا بطور خلاصه rspec ، انجام می پذیرد. rspec بیانگر فاصله زمانی است که در طی آن منابع بخصوص تخصیص می یابند. این منابع شامل فضای دیسک، توصیف گرهای فایل، پهنهای باند شبکه داخل محدوده<sup>۴۰۰</sup> و خارج محدوده<sup>۴۰۱</sup> ، نقاط پایانی

<sup>399</sup> slice  
<sup>400</sup> Inbound

سطح انتقال، حافظه اصلی، و به کارگیری CPU می باشد. *rspec* از طریق یک شناسه ۱۲۸ بیتی منحصر به فرد جهانی به نام توانایی منبع<sup>۴۰۲</sup> (*rcap*) شناسایی می شود. مدیر گره با در اختیار داشتن *rspec* می تواند *rcap* مربوطه را در جدول محلی جستجو کند.

منابع به برش ها بسته هستند. به بیان دیگر، برای استفاده از منابع باید یک برش ایجاد شود. هر برش با یک ارائه کننده سرویس<sup>۴۰۳</sup> - موجودیتی که روی PlanetLab حساب دارد- مرتبط می باشد. بنابراین، هریک از برشها را می توان بوسیله جفت (*principal-id* ، *slice-tag*) شناسایی نمود، که در آن *principal-id* بیانگر ارائه کننده و *slice-tag* ، شناسه انتخاب شده توسط ارائه کننده است. برای ایجاد یک برش جدید، گره باید یک سرویس ایجاد برش<sup>۴۰۴</sup> (*SCS*) را اجرا کند . این سرویس هم ضمن تماس با مدیر گره از او درخواست ایجاد یک v서ور و تخصیص منابع را خواهد داشت. نمی توان مستقیماً از روی شبکه با خود مدیر گره تماس برقرار کرده و به این ترتیب مدیر گروه فقط روی مدیریت منبع محلی تمرکز می کند. *SCS* هم به نوبه خود درخواستهای ایجاد برش از جانب هر کسی را نمی پذیرد، بلکه فقط برخی از مسئولین برش<sup>۴۰۵</sup> مجاز به درخواست ایجاد برش هستند. هریک از مسئولین برش ها حق دسترسی به مجموعه ای از گره ها را دارند. در ساده ترین مدل، فقط یک مسئول برش وجود دارد و او مجاز به درخواست ایجاد برش روی تمامی گره هاست.

برای تکمیل بحث خود در نظر می گیریم که ارائه دهنده سرویس ضمن تماس با مسئول برش ، از او درخواست ایجاد برش در مجموعه ای از گره ها را خواهد کرد. ارائه دهنده سرویس هم ، مثلاً به این دلیل که قبلاً تأیید هویت شده و بعنوان کاربر PlanetLab ثبت شده بود، برای مسئول برش شناخته شده خواهد بود. در عمل، کاربران PlanetLab بوسیله یک سرویس براساس وب با مسئول برش تماس برقرار می کنند. جزئیات بیشتر در این رابطه را در مرجع ( Chun and Spalink ۲۰۰۳ ) مطالعه کنید.

از این روال می توان دریافت که مدیریت PlanetLab از طریق میانجی ها انجام می شود. مسئولین برش یکی از انواع مهم این میانجی ها را تشکیل می دهند. این مسئولین، اعتبارنامه هایی<sup>۴۰۶</sup> را در گره ها برای ایجاد برش دریافت کرده اند. کسب این اعتبارنامه ها به صورت خارج باند و اساساً از طریق تماس با مدیران اجرایی سیستم در سایتهای مختلف صورت می پذیرد . ناگفته پیداست که این فرآیند زمانی، بوسیله کاربران نهایی ( یا به زبان PlanetLab ، ارائه دهنده گان سرویس) انجام نخواهد شد.

<sup>401</sup> outbound

<sup>402</sup> resource capability

<sup>403</sup> service provider

<sup>404</sup> slice creation service

<sup>405</sup> slice authorities

<sup>406</sup> credentials

علاوه بر مسئولین برش، مسئولین مدیریت<sup>۴۰۷</sup> نیز وجود دارند. در حالیکه مسئول برش فقط روی مدیریت برشها تمرکز می کند، مسئول مدیریت مراقب از گره ها را بر عهده دارد، و بخصوص تضمین می کند که گره های تحت حاکمیت او نرم افزار اصلی PlanetLab را اجرا کرده وتابع قوانین وضع شده توسط PlanetLab باشند. ارائه دهنده سرویس اطمینان دارند که گره های ارائه شده به وسیله مسئول مدیریت، رفتار مناسبی خواهند داشت.

### شکل ۳-۱۶- روابط مدیریتی بین موجودیت های مختلف PlanetLab (داخل شکل: مالک گره، مسئول مدیریت، ارائه دهنده سرویس، مسئول برش).

سازمان فوق الذکر منجر به ساختار مدیریتی شکل ۳-۱۶ می شود که در مرجع (Peterson، ۲۰۰۵، و گروه همکاران) از نقطه نظر روابط اعتمادی تشریح شده است. این روابط به شرح زیر است:

۱. مالک گره مفروض، گره خود را تحت تسلط مسئول مدیریت قرار داده و در صورت نیاز، به کارگیری راهم محدود می کند.

۲. مسئول مدیریت نرم افزار لازم را برای افزودن گره به PlanetLab ارائه می کند.

۳. ارائه دهنده سرویس ضمن اعتماد به این که مسئول مدیریت گره هایی با رفتار مناسب ایجاد می کند، در آن ثبت نام می نماید.

۴. ارائه دهنده سرویس برای ایجاد برش روی مجموعه ای از گره ها، با مسئول برش تماس برقرار می کند.

۵. مسئول برش بایستی ارائه دهنده سرویس را تأیید هویت نماید.

۶. مالک گره جهت ایجاد برش، سرویس ایجاد برش را برای مسئول برش ایجاد نموده و مدیریت منبع را به مسئول برش واگذار می کند.

۷. مسئول مدیریت ایجاد برشها را به مسئول برش واگذار می کند

این روابط بیانگر مشکل تفویض اختیار به گره ها به صورتی کنترل شده است تا مالک گره بتواند از وجود مدیریتی ایمن و بقاعده اطمینان داشته باشد . مسئله دوم که باید مورد بررسی قرار گیرد، مسئله نظارت است. لازم است با اتخاذ شیوه ای واحد، به کاربران اجازه داده شود تا بر رفتار برنامه های خود در داخل برشی خاص نظارت داشته باشند.

PlanetLab از شیوه ساده ای تبعیت می کند. به این ترتیب که هریک از گره ها مجهز به مجموعه ای از حسگرها بوده و هر حسگر قادر به گزارش دهی اطلاعاتی از قبیل کارکرد CPU، فعالیت دیسک و غیره می باشد. حسگرها ممکن است بسیار پیچیده باشند، اما مسئله مهم آن است که همواره گزارشات خود را براساس هرگره ارائه می دهند. این اطلاعات بوسیله یک سرور وب در اختیار دیگران قرار می

<sup>407</sup> management authorities

گیرد، بطوریکه هریک از حسگرها از طریق درخواستهای ساده HTTP قابل دسترسی هستند (۲۰۰۴، Bavier و گروه همکاران).

هرچند این شیوه نظارتی هنوز در مراحل اولیه خود بسر می برد ، اما می تواند به عنوان مبنای برای طرح های پیشرفته نظارتی مدنظر قرار گیرد . بعنوان مثال، دلیلی وجود ندارد که نتوان از Astrolabe که در فصل ۲ مورد بحث قرار گرفت، جهت خواندن تجمعی حسگر متراکم در بین چندین گره استفاده نشود.

بعنوان سومین و آخرین نکته در مورد مسائل مدیریتی، یعنی حفاظت از برنامه ها در برابر یکدیگر، باید گفت که PlanetLab از سرورهای مجازی Linux ( که به نام ۷ سرورها خوانده می شوند) جهت جدا سازی بین برش ها استفاده می کند. همانطور که قبلًا هم گفته شد، ایده اصلی در یک ۷ سرور، اجرای برنامه های کاربردی در محیط خودشان است. محیط شامل تمامی فایل هایی است که معمولاً در داخل یک ماشین به اشتراک گذاشته می شوند. دستیابی به این جداسازی تقریباً آسان و از طریق فرمان UNIX chroot امکان پذیر است ؛ فرمان مذکور در عمل باعث تغییر ریشه سیستم فایل از محل جستجوی فایلها توسط برنامه های کاربردی می شود . فقط ابرکاربرها<sup>۴۰۸</sup> قادر به اجرای chroot هستند.

البته هنوز ناگفته ها بسیار است. سرورهای مجازی Linux علاوه بر جداسازی سیستم های فایل، اطلاعات معمولاً مشترک در مورد فرآیندها، آدرسهای شبکه ای، کارکرد حافظه و غیره را هم جدا می سازند. در نتیجه، هر ماشین فیزیکی مفروض واقعاً به چندین واحد تقسیم شده و هر واحد در رابطه با یک محیط تمام عیار Linux است که مجزا از بخش های دیگر است. مروری بر سرورهای مجازی Linux را در مرجع (۲۰۰۵) Potzl و گروه همکاران مطالعه کنید.

---

<sup>408</sup> superuser

### ۴۰۹-۳-۵- مهاجرت کد

تا به اینجا، ما بیشتر برروی سیستم‌های توزیعی تمرکز کرده‌ایم که ارتباطات آنها منحصر به عبور داده هاست. با این وجود، مواردی هم وجود دارند که عبوردادن برنامه‌ها حتی در حین اجرای آنها هم، باعث ساده سازی طرح سیستم‌های توزیعی خواهد شد. در این بخش، نگاه دقیقی به مفهوم واقعی مهاجرت کد خواهیم داشت. کار را با بررسی روش‌های مختلف مهاجرت کد آغاز کرده و سپس، به سراغ منابع محلی مورد استفاده در برنامه‌های مهاجرتی خواهیم رفت. یک مسئله مشکل مهاجرت کد در سیستم‌های ناهمگن است که در همین بخش بررسی می‌شود.

#### ۱-۳-۵- روش‌های مهاجرت کد

پیش از ورود به مبحث انواع مختلف مهاجرت کد، ابتدا دلایل سودمند بودن مهاجرت کد را بررسی می‌کنیم.

#### دلایل مهاجرت کد

سابقاً، مهاجرت کد در سیستم‌های توزیعی به شکل مهاجرت فرآیند<sup>۴۱۰</sup> انجام می‌شد و طی آن، کل فرآیند از یک ماشین به ماشین دیگر انتقال می‌یافتد (۲۰۰۰، Milojicic و گروه همکاران). انتقال یک برنامه در حال اجرا به ماشین دیگر کاری هزینه برد و دقیق است و بهتر است دلیل خوبی برای این کار وجود داشته باشد. دلیل اصلی همواره بهبود کارآیی بوده است. ایده اصلی این است که می‌توان با انتقال فرآیندها از ماشینهای دارای بار زیاد به ماشینهای کم بار، کارآیی کلی سیستم را افزایش داد. بار را غالباً به صورت طول صفحه پردازنده یا بهره وری از پردازندۀ تعریف می‌کنند، اما شاخصه‌های کارآیی دیگری هم می‌توان به این تعریف اضافه کرد.

الگوریتم‌های توزیع بار- که تصمیمات مربوط به تخصیص و توزیع مجدد کارها با توجه به مجموعه پردازنده‌ها براساس آن گرفته می‌شود- نقش مهمی را در سیستم‌های محاسباتی سنگین ایفا می‌کنند. با این وجود، در بسیاری از سیستم‌های مدرن توزیعی، بهینه سازی ظرفیت محاسباتی به اندازه مسائلی از قبیل به حداقل رساندن ارتباط، مهم محسوب نمی‌شود. همچنین، به دلیل ناهمگن بودن بسترهای زیربنایی و شبکه‌های کامپیوتری، بهبود کارآیی از طریق مهاجرت کد به جای مدل‌های ریاضی، غالباً براساس استدلالهای کمی استوار است.

بعنوان نمونه، سیستم مشتری-سروری را در نظر بگیرید که سرور آن، مدیریت پایگاه داده ای عظیمی را برعهده داشته باشد. چنانچه برنامه کاربردی مشتری نیاز به انجام عملیات پایگاه داده ای متعدد بر روی مقادیر بزرگ داده ای داشته باشد، شاید بهتر باشد که بخشی از برنامه کاربردی مشتری به سرور

<sup>409</sup> code migration

<sup>410</sup> process migration

انتقال یافته و فقط نتایج بر روی شبکه ارسال شود. در غیراینصورت، ممکن است شبکه با سیل عظیمی از داده های در حال تبادل از سرور به مشتری مواجه شود. در این حالت، مهاجرت کد بر اساس این فرضیه صورت خواهد پذیرفت که بهتر است داده ها در مجاورت محل استقرار خودشان پردازش شوند. از همین استدلال می توان برای مهاجرت بخشهایی از سرور به مشتری استفاده نمود. عنوان مثال، در بسیاری از برنامه های کاربردی تعاملی پایگاه داده ای، مشتریان باید فرمهایی را پرکنند که بعداً بصورت مجموعه ای از عملیات پایگاه داده ای ترجمه خواهند شد. پردازش این فرم در طرف مشتری و ارسال فرم تکمیل شده برای سرور، ممکن است مانع از عبور تعداد نسبتاً زیادی از پیامهای کوچک در شبکه شود. نتیجه آنکه مشتری با کارآیی بهتری مواجه شده و سرور هم مدت زمان کمتری را صرف پردازش فرم و ارتباطات خواهد کرد.

علاوه، پشتیبانی از مهاجرت کد می تواند با بهره گیری از موازی گرایی<sup>۴۱۱</sup>، اما بدون در نظر گرفتن دقتهای معمول در واسطه با برنامه سازی موازی، به بهبود کارآیی کمک کند. عنوان نمونه می توان به جستجوی اطلاعات در وب اشاره کرد. تقریباً به راحتی می توان پرس و جوی<sup>۴۱۲</sup> جستجو را به شکل برنامه سیار کوچکی به نام نماینده سیار<sup>۴۱۳</sup>، که می تواند از یک سایت به سایت دیگر حرکت کند اجرا نمود. با تهیه چندین کپی از این نوع برنامه ها و ارسال آنها به سایتها مختلف، ممکن است بتوان در مقایسه با شیوه استفاده از فقط یک برنامه به طور خطی سرعت را افزایش داد.

علاوه بر بهبود کارآیی، دلایل دیگری هم برای پشتیبانی از ایده مهاجرت کد می توان ارائه کرد که مهمترین آن، مسئله انعطاف پذیری است. روش متعارف در ساخت برنامه های کاربردی توزیعی، تقسیم برنامه کاربردی به چندین بخش، و تصمیم گیری پیش از اجرا در مورد محل مناسب اجرای هر یک از بخشهاست. این روش، عنوان مثال، منجر به برنامه های کاربردی مشتری-سرور چندطبقه ای مختلفی شده که در فصل دو مورد بحث قرار گرفت.

چنانچه کد بتواند بین چندین ماشین جابجا شود، پیکره بندی پویای سیستم های توزیعی هم امکانپذیر می شود. عنوان مثال، فرض کنید که یک سرور مفروض واسط استانداردی به یک سیستم فایل را پیاده سازی کرده است. برای فراهم آوردن امکان دسترسی مشتریان دور به سیستم فایل، سرور از یک پروتکل اختصاصی<sup>۴۱۴</sup> استفاده می کند. معمولاً، پیاده سازی واسط سیستم فایل در طرف مشتری، که براساس این پروتکل استوار است، نیازمند برقراری پیوند با برنامه کاربردی مشتری است. در این روش لازم است که نرم افزار در زمان ایجاد برنامه کاربردی مشتری، موجود و در دسترس مشتری باشد.

یک روش جایگزین آن است که سرور پیاده سازی مورد نیاز مشتری را فقط زمانی که مشتری نیازمند آن است ارائه نماید، یعنی زمانی که مشتری به سرور پیوند می خورد. در این زمان، مشتری به گونه ای

<sup>411</sup> parallelism

<sup>412</sup> query

<sup>413</sup> mobile agent

<sup>414</sup> proprietary protocol

پویا این پیاده سازی را دانلود می کند، مراحل اولیه مورد نیاز را می گذراند، و نهایتاً سرور را فرامی خواند. این مطلب در شکل ۳-۱۷ نشان داده شده است. این مدل انتقال پویایی کد از یک سایت دور مسلماً نیازمند یک پروتکل استاندارد برای دانلود کد و گذراندن مراحل آغازین است. بعلاوه باید بتوان کد دانلودشده را روی ماشین مشتری اجرا نمود. راه حلهای مختلف برای اینکار در این فصل و در فصلهای بعد بررسی می شوند.

**شکل ۳-۱۷ - اصل کلی پیکر بندی پویای مشتری جهت برقراری ارتباط با سرور.** مشتری ابتدا نرم افزار لازمه را واکشی می نماید و سپس سرور را فرا می خواند) داخل شکل: کد ویژه سرویس طرف مشتری، مشتری، سرور، ۱- مشتری کد را واکشی می کند، ۲- مشتری و سرور با هم ارتباط برقرار می کنند، مخزن کد).

یکی از مزایای مهم مدل دانلود پویای نرم افزار طرف مشتری آن است که دیگر لازم نیست مشتریان برای صحبت با سرورها تمام نرم افزار را ازپیش نصب شده داشته باشند. بلکه می توان در صورت لزوم نرم افزار را انتقال داده و سپس در زمانی که به آن نیازی نباشد آنرا دور ریخت. یک مزیت دیگر این است که در صورت استاندارد بودن واسطه ها ، می توان پروتکل مشتری- سرور و پیاده سازی آنرا به هر تعداد که مورد نیاز باشد تغییر داد. این تغییرات تأثیری بر برنامه های کاربردی مشتری که متکی به سرور هستند نخواهد داشت. البته معایبی هم دارد که از مهمترین آنها می توان به مشکلات امنیتی اشاره کرد که در فصل ۹ بحث می شود. اطمینان کورکورانه از اینکه کد دانلود شده در صورت دسترسی به هارد دیسک محافظت نشده شما، فقط واسطه اعلان شده را اجرا خواهد کرد و ناب ترین بخشهای آنرا به ناکجا آباد نخواهد فرستاد، ممکن است همیشه نتایج چندان مطلوبی هم به بار نیاورد.

#### مدلهای مهاجرت کد

هرچند مهاجرت کد به معنای جابجایی کد در بین ماشینهای است ، این اصطلاح در واقع حوزه بزرگتری را شامل می شود. بطور معمول ، ارتباطات در سیستم های توزیعی به تبادل داده ها بین فرآیندها می پردازد. مهاجرت کد در گستره ترین حالت، به موضوع جابجایی برنامه ها در بین ماشینهای با هدف اجرای این برنامه ها در مقصد انجام می پذیرد. در برخی موارد از جمله مهاجرت فرآیند، وضعیت اجرای یک برنامه، سیگنالهای در انتظار<sup>۴۱۵</sup> و بخشهای دیگر محیط هم باید انتقال داده شوند.

برای درک بهتر مدل‌های مهاجرت کد، از چارچوب مشروح در مرجع (۱۹۹۸) Fuggetta و گروه همکاران استفاده می کنیم. براساس این چارچوب، هر فرآیند از سه قطعه تشکیل می شود. قطعه کد<sup>۴۱۶</sup> بخش حاوی مجموعه دستورالعملهای تشکیل دهنده برنامه در حال اجرا است. قطعه منبع<sup>۴۱۷</sup> شامل مرجع ها به منابع خارجی مورد نیاز در فرآیند از قبیل فایلها، چاپگرهای، دستگاهها، فرآیندهای دیگر و از

<sup>415</sup> pending signals

<sup>416</sup> Code Segment

<sup>417</sup> resource segment

این قبیل است. قطعه آخر یعنی قطعه اجرا<sup>۴۱۸</sup> هم جهت ذخیره سازی وضعیت اجرای فعلی فرآیند از جمله داده های خصوصی<sup>۴۱۹</sup>، پشته و، البته، شمارشگر برنامه استفاده می شود.

مهاجرت کد در حالت حداقل خود فقط نوعی **جابجایی ضعیف**<sup>۴۲۰</sup> ارائه می کند. در این مدل، می توان فقط قطعه کد و در صورت لزوم، برخی داده های آغازین<sup>۴۲۱</sup> را انتقال داد. یکی از ویژگی های بارز **جابجایی ضعیف** آن است که برنامه انتقال یافته همواره از یکی از چندین موقعیت آغازکننده از پیش تعریف شده آغاز می شود. عنوان نمونه می توان به کوچک برنامه های<sup>۴۲۲</sup> Java اشاره کرد که همواره کار اجرا را از ابتدا آغاز می کنند. مزیت این رویکرد در سادگی آن است. **جابجایی ضعیف** نیازمند آن است که فقط ماشین هدف قادر به اجرای کد باشد، و نهایتاً به کد قابل **جابجایی منجر** می شود. به هنگام بحث در مورد مهاجرت در سیستم های ناهمگن مجدداً به این موضوع بازخواهیم گشت.

برخلاف **جابجایی ضعیف**، در سیستم هایی که از **جابجایی قوی**<sup>۴۲۳</sup> پشتیبانی می کنند، قطعه اجرا هم قابل انتقال است. ویژگی بارز **جابجایی قوی** آن است که فرآیند در حال اجرا را می توان متوقف ساخته، به ماشین دیگری انتقال داد و سپس ، کار اجرا را از همانجا که متوقف شده، مجدداً از سر گرفت. مشخص است که هرچند **جابجایی قوی** بسیار عمومی تر از **جابجایی ضعیف** است، اما پیاده سازی آن بسیار مشکل است.

صرف نظر از **جابجایی ضعیف** و **قوی**، باید بین مهاجرت آغاز شده از فرستنده و آغاز شده از گیرنده هم تمایز قائل شد. در حالت مهاجرت آغاز شده از فرستنده<sup>۴۲۴</sup>، مهاجرت از ماشینی آغاز می شود که کد در آن اقامت داشته یا در حال اجراست. نوعاً، مهاجرت آغاز شده از فرستنده در هنگام آپلود کردن برنامه ها به کامپیوتر سرور انجام می شود. مثال دیگر، ارسال برنامه جستجو از طریق اینترنت به سرور پایگاه داده ای و بجهت انجام پرس و جو در آن سرور است. در مهاجرت آغاز شده از گیرنده<sup>۴۲۵</sup>، اقدام برای مهاجرت کد بوسیله ماشین هدف انجام می شود. کوچک برنامه های Java نمونه ای از این روش است.

مهاجرت آغاز شده از گیرنده ساده تر از مهاجرت آغاز شده از فرستنده است. در بسیاری از موارد، مهاجرت کد بین مشتری و سرور انجام شده و مشتری برای مهاجرت آغازگر است. آپلود کردن این کد به سرور، که در مهاجرت آغاز شده از فرستنده صورت می پذیرد، غالباً مستلزم آن است که مشتری قبلاً در آن سرور ثبت<sup>۴۲۶</sup> و تأیید<sup>۴۲۷</sup> شده باشد. به بیان دیگر، از آنجاییکه مشتری به احتمال قوی خواهان

<sup>418</sup> execution segment

<sup>419</sup> private data

<sup>420</sup> weak mobility

<sup>421</sup> initialization data

<sup>422</sup> applets

<sup>423</sup> strong mobility

<sup>424</sup> sender-initiated

<sup>425</sup> receiver-initiated

<sup>426</sup> register

دسترسی به منابع سرور از جمله دیسک آن خواهد بود، لازم است که سرور تمامی مشتری های خود را بشناسد . محافظت از چنین منابعی ضروری است. بالعکس، دانلود کردن کد همانند حالت آغاز شده از گیرنده ، ممکن است بطور ناشناس انجام شود. بعلاوه، سرور معمولاً علاقه ای به منابع مشتری نداشته و مهاجرت کد به سمت مشتری فقط با هدف بهبود کارآیی طرف مشتری انجام می شود. از اینرو ، فقط تعداد معادلی از منابع از قبیل اتصالات شبکه و حافظه نیازمند محافظت هستند. در فصل ۹ مفصلًاً راجع به اینمی در مهاجرت کد بحث خواهیم کرد.

در حالت جابجایی ضعیف، بین اجرای کد مهاجرت کرده توسط فرآیند هدف و یا ایجاد فرآیند دیگر برای اجرای کد تفاوت وجود دارد. بعنوان مثال، کوچک برنامه های Java صرفاً بوسیله جستجوگر وب (فرآیند هدف) دانلود شده و در فضای آدرس جستجوگر وب اجرا می شوند. مزیت روش این است که دیگر نیازی به ایجاد فرآیند دیگری نبوده و بدین ترتیب از ارتباطات در ماشین هدف جلوگیری می شود. اما مهمترین نقطه ضعف این است که فرآیند هدف باستی در برابر اجراهای سهولی یا مغرضانه کد محافظت شود. یک راه حل ساده آن است که اجازه دهیم تا سیستم عامل با ایجاد فرآیند جداگانه ای جهت اجرای کد مهاجرت کرده، اقدام به مراقبت کند. توجه داشته باشید که این روش مشکلات دسترسی به منابع را حل نمی کند. بنابراین موضوع دسترسی هنوز هم نیازمند بحث و بررسی بیشتر است .

در عوض جا به جایی یک فرآیند در حال اجرا (جا به جایی فرآیند)، جا به جایی قوی را می توان با روش تقلید از دور<sup>۴۲۸</sup> نیز پشتیبانی نمود. برخلاف مهاجرت فرآیند ، در روش تقلید، کپی دقیقی از فرآیند اصلی ایجاد می شود که در همان زمان بر روی ماشین دیگری در حال اجراست. فرآیند مقلد به طور موازی با فرآیند اصلی اجرا می شود. در سیستم های UNIX، تقلید از راه دور از طریق ایجاد فرآیند فرزند و ادامه یافتن آن برروی یک ماشین دور انجام می شود. مزیت مدل تقلید این است که بسیار شبیه آن چیزی است که هم اکنون در بسیاری از برنامه های کاربردی مورد استفاده قرار می گیرد. تنها تفاوت این است که فرآیند مقلد روی ماشین دیگری اجرا می شود. از این نقطه نظر، مهاجرت به روش تقلید روش ساده ای برای بهبود شفافیت توزیع محسوب می شود. روشهای مختلف مهاجرت کد در شکل ۳-۱۸ خلاصه شده است.

**شکل ۳-۱۸-روشهای مختلف مهاجرت کد**( داخل شکل: مکانیزم جابجایی، جابجایی ضعیف، جابجایی آغاز شده از فرستنده، اجرا در فرآیند هدف، اجرا در یک فرآیند جداگانه، جابجایی آغاز شده از گیرنده، اجرا در فرآیند هدف، اجرا در یک فرآیند جداگانه، جابجایی قوی، جابجایی آغاز شده از فرستنده، مهاجرت فرآیند، تقلید فرآیند، جابجایی آغاز شده از گیرنده، مهاجرت فرآیند، تقلید فرآیند).

<sup>427</sup> authenticate

<sup>428</sup> Remote Cloning

### ۲-۵-۳- مهاجرت و منابع محلی<sup>۴۲۹</sup>

تا به اینجا، در مورد مهاجرت کد و قطعه اجرا صحبت کرده ایم. قطعه منبع نیازمند توجه ویژه ای است. آنچه غالباً مهاجرت کد را تبدیل به امری بسیار مشکل می کند این است که نمی توان قطعه منبع را براحتی و بدون هیچ تغییری همراه با قطعه های دیگر انتقال داد. عنوان مثال، فرآیندی را فرض کنید که مرجعی<sup>۴۳۰</sup> به یک درگاه TCP خاص دارد و از طریق آن با فرآیندهای (از راه دور) دیگر ارتباط برقرار می کند. این مرجع در قطعه منبع نگه داشته می شود. در صورتیکه فرآیند به محل دیگری انتقال پیدا کند، باید درگاه قبلی را رها کرده و درگاه جدیدی را در محل مقصد درخواست کند. در مواردی دیگر، انتقال مرجع لزوماً مشکل‌ساز نخواهد بود. عنوان مثال، ارجاع به یک فایل از طریق یک URL مطلق<sup>۴۳۱</sup>، بدون در نظر گرفتن ماشینی که فرآیند دارای URL در آن اقامت دارد، همچنان معابر خواهد ماند. برای درک اثرات مهاجرت کد بر قطعه منبع ، (۱۹۹۸) Fuggetta و گروه همکاران سه نوع پیوند<sup>۴۳۲</sup> فرآیند به منبع را شناسایی کرده اند. قوی ترین پیوند هنگامی است که یک فرآیند از طریق شناسه<sup>۴۳۳</sup> خود به منبع ارجاع کند. در این حالت، فرآیند دقیقاً نیازمند همان منبع ارجاع داده شده خواهد بود و نه چیز دیگری. عنوان نمونه ای از این پیوند توسط شناسه<sup>۴۳۴</sup> می توان به مواردی اشاره کرد که فرآیند ، جهت ارجاع به یک وب سایت خاص از URL استفاده کند یا هنگامیکه بوسیله آدرس اینترنتی سرور به سرور FTP آن ارجاع می کند. بر پایه همین دیدگاه ارجاعات به نقاط پایانی ارتباطی محلی نمونه دیگری از پیوند توسط شناسه محسوب می شود.

یکی از حالات ضعیف تر پیوند فرآیند به منبع هنگامی است که فقط مقدار منبع مورد نیاز باشد. در این حالت، اجرای فرآیند دستخوش تغییری نخواهد شد اگر منبع دیگری مقداری برابر با مقدار منبع قبلی ارائه کند. یکی از نمونه های پیوند توسط مقدار<sup>۴۳۴</sup> هنگامی است که برنامه بر کتابخانه های استانداردی تکیه می کند که مثلاً در برنامه نویسی در C یا Java مورد استفاده قرار می گیرند. این دست کتابخانه ها باستی همواره بصورت محلی قابل دسترسی باشند ، اما ممکن است محل دقیق آنها در سیستم فایل محلی در سایتها مختلف متفاوت باشد. برای اجرای مناسب و موفقیت آمیز فرآیند، نه خود فایلها، بلکه محتوای آنها اهمیت دارد.

ضعیف ترین و آخرین شکل پیوند هنگامی است که فرآیند فقط نیازمند منبعی از نوع مشخصی باشد. از نمونه های این پیوند توسط نوع<sup>۴۳۵</sup> می توان به ارجاعات به دستگاههای محلی از قبیل صفحه نمایش ها، چاپگرهای از این قبیل اشاره کرد.

<sup>429</sup> local resources

<sup>430</sup> Reference

<sup>431</sup> absolute

<sup>432</sup> Identifier

<sup>433</sup> binding by identifier

<sup>434</sup> binding by value

<sup>435</sup> binding by type

برای مهاجرت کد غالباً لازم است که ارجاعات به منابع را تغییر دهیم، اما نمی‌توانیم بر چگونگی پیوند فرآیند به منبع تأثیر بگذاریم. چراکه و چگونگی تغییر ارجاع به یک منبع بستگی به آن دارد که آیا می‌توان منبع را بهمراه کد به ماشین هدف انتقال داد یا خیر. به بیان دقیق‌تر، باید پیوند منبع به ماشین را در نظر گرفته و بین موارد زیر تمایز قائل شویم. منابع نابسته<sup>۴۳۶</sup> را می‌توان براحتی در بین ماشینهای مختلف جابجا کرده و نوعاً فایل‌های (داده‌ای) مرتبط با برنامه باید مهاجرت کنند. هرچند امکان انتقال یا کپی کردن منبع بسته شده<sup>۴۳۷</sup> وجود دارد، این کار هزینه نسبتاً زیادی دارد. بعنوان نمونه‌های منابع بسته شده می‌توان به پایگاه‌های داده‌ای محلی و وب سایتها کامل اشاره کرد. هرچند این منابع، در تئوری، به ماشین فعلی خود وابسته نیستند، غالباً انتقال آنها به محیط دیگر امکان پذیر نیست. منابع ثابت<sup>۴۳۸</sup> هم به شدت به ماشین یا محیط خاصی متصل بوده و بنابراین قابل انتقال نیستند. این منابع غالباً دستگاه‌های محلی هستند. مثال دیگری از منابع ثابت نقاط پایانی ارتباط محلی<sup>۴۳۹</sup> هستند. ترکیب سه نوع پیوند فرآیند به منبع با سه نوع پیوند منبع به ماشین منجر به نه صورت ترکیبی می‌شود که باید در مهاجرت کد مدنظر قرار گیرند. این نه حالت در شکل ۱۹-۳ نمایش داده شده است.

پیوند منبع به ماشین				
		نابسته	بسته	ثبت
پیوند فرآیند به منبع	توسط شناسه	MV (or GR)	GR (or MV)	GR
	توسط مقدار	CP (or MV,GR)	GR (or CP)	GR
	توسط نوع	RB (or MV,GR)	RB (or GR,CP)	RB (or GR)

ایجاد یک مرجع جهانی برای تمام سیستم	GR (Global Reference)
جابجایی منبع	MV (Move)
کپی کردن مقدار منبع	CP (Copy)
الحاق مجدد فرآیند به منبع قابل دسترسی به صورت محلی	RB (Rebind)
شکل ۱۹-۳- اقداماتی که باید با توجه به ارجاعات به منابع محلی در مهاجرت کد به ماشین دیگر انجام شود.	

ابتدا باید نگاهی به حالات مختلف پیوند فرآیند به منبع توسط شناسه داشته باشیم. در صورت نابسته بودن منبع، معمولاً بهترین گزینه جابجایی آن بوسیله کد در حال مهاجرت است. با این وجود، هنگامیکه منبع با فرآیندهای دیگر مشترک باشد، یک راه حل ایجاد یک مرجع جهانی<sup>۴۴۰</sup> یعنی مرجعی

<sup>۴۳۶</sup> unattached resources

<sup>۴۳۷</sup> fastened resource

<sup>۴۳۸</sup> fixed resources

<sup>۴۳۹</sup> local communication end points

<sup>۴۴۰</sup> global reference

است که بتواند از مرزهای ماشین عبور کند . یکی از نمونه های چنین مراجعی URL<sup>۴۴۱</sup> ها هستند. در صورتیکه منبع بسته شده یا ثابت باشد، بهترین راه حل بازهم ایجاد مرجع جهانی خواهد بود. باید بدایم که ایجاد مرجع جهانی ممکن است مستلزم کارهایی بیش از استفاده از URLها بوده و استفاده از چنین مرجعی می تواند بسیار هزینه بر باشد. عنوان مثال برنامه ای را فرض کنید که تصاویر با کیفیت بالایی را برای یک ایستگاه کاری مخصوص چندسانه ای تولید می کند. تولید تصاویر با کیفیت بالا به صورت بی درنگ نیازمند کار محاسباتی زیادی بوده و به همین دلیل ممکن است برنامه به یک سرور محاسباتی کارآ انتقال داده شود. برقراری مرجع جهانی برای ایستگاه کاری چندسانه ای به معنای ایجاد یک مسیر ارتباطی بین سرور محاسباتی و ایستگاه کاری است. بعلاوه، سرور و ایستگاه کاری هردو نیازمند انجام پردازش های زیادی برای دست یابی به پنهانی باند مورد نیاز جهت انتقال تصاویر هستند. در نتیجه، ممکن است انتقال برنامه به سرور محاسباتی، به دلیل هزینه بر بودن زیاد مرجع جهانی، چندان هم ایده خوبی محسوب نشود .

عنوان نمونه دیگری از مواردی که برقراری مرجع جهانی همواره چندان هم ساده نیست، می توان به مهاجرت فرآیندی اشاره کرد که از نقطه پایانی ارتباط محلی استفاده می کند. در این حالت، ما با منبع ثابتی مواجه هستیم که فرآیند توسط شناسه به آن پیوند دارد. اساساً دو راه حل در این واسطه وجود دارد. یک راه حل آن است که فرآیند پس از مهاجرت، ضمن ایجاد اتصال با ماشین منبع ، فرآیند جداگانه ای را در ماشین منبع نصب کرده و بكمک آن براحتی تمامی پیامهای ورودی به سمت فرآیند مهاجرت کرده پیش برده شود. نقطه ضعف عمدۀ این روش آن است که در صورتی که ماشین منبع دچار اشکال شود، ارتباط با فرآیند مهاجرت کرده ممکن است قطع شود. راه حل دیگر آن است که تمامی فرآیندهایی که قبلاً با فرآیند در حال مهاجرت ارتباط برقرار کرده بودند ، مرجع جهانی خود را تغییر داده و پیامهای جدید را به نقطه پایانی ارتباطی جدید در ماشین هدف ارسال کنند.

در مبحث پیوند به توسط مقدار با شکل متفاوتی مواجه هستیم. ابتدا یک منبع ثابت را در نظر بگیرید. ترکیب این منبع ثابت و پیوند به توسط مقدار هنگامی رخ می دهد که فرآیندی فرض کند که از حافظه مشترک بین فرآیندها می تواند استفاده کند. ایجاد مرجع جهانی در چنین حالتی به این معنای لزوم پیاده سازی شکل توزیعی از حافظه اشتراکی است . در بسیاری از موارد، این راه حل واقعاً قابل اجرا و مؤثر نیست.

منابع بسته شده که براساس مقدارشان ارجاع داده می شوند، نوعاً کتابخانه های زمان اجرا<sup>۴۴۲</sup> هستند . معمولاً، کپی های این دست منابع روی ماشین هدف قابل دستیابی بوده و یا باستی به نحوی تا پیش از مهاجرت کد، کپی شوند. درصورت زیاد بودن حجم داده های کپی شونده، مثلاً در تهیه فرهنگهای لغت و واژه یابها در سیستم های پردازش متن، ایجاد مرجع جهانی گزینه بهتری خواهد بود.

<sup>441</sup> Universal Resource Locator

<sup>442</sup> runtime

ساده ترین حالت مربوط به منابع نابسته است. به استثنای مواردی که منبع در بین چند فرآیند مشترک باشد، بهترین راه حل کی کردن ( یا انتقال) منبع به مقصد جدید می باشد. اما در حالت استثناء گفته شده ، ایجاد مرجع جهانی تنها گزینه محسوب می شود.

آخرین حالت مربوط به پیوند به توسط نوع است. غیر از پیوند منبع به ماشین، راه حل مشخص پیوند مجدد فرآیند به یک منبع قابل دستیابی به صورت محلی از همان نوع است. تنها در صورت نبود چنین منبعی است که باید منبع اصلی را به مقصد جدید کپی کرده یا انتقال داده، و یا اقدام به ایجاد مرجع جهانی نمود.

### ۴۴۳-۳-۵-۳- مهاجرت در سیستم های ناهمگن

تا به اینجا، به طور ضمنی فرض کرده ایم که کد مهاجرت کرده در ماشین هدف به راحتی قابل اجراست . این فرض در بحث در مورد سیستم های همگن<sup>۴۴۴</sup> صدق می کند. اما در مجموع باید گفت که سیستم های توزیعی براساس مجموعه ناهمگنی از بسترها استوار هستند که هریک دارای سیستم عامل و معماری ماشین مختص به خود می باشد. در چنین سیستم هایی، موفقیت مهاجرت مستلزم پشتیبانی از تمامی بسترهاست؛ به این معنا که، بتوان قطعه کد را روی هریک از بسترها اجرا نمود. همچنین باید از وجود قطعه اجرا در هریک از بسترها اطمینان داشت.

مشکلات ناشی از ناهمگن بودن در بسیاری از موارد مشابه مشکلات مربوط به قابلیت جابجایی است . بنابراین تعجبی ندارد اگر بگوییم که راه حل‌های آنها هم بسیار مشابه هستند. بعنوان مثال، در پایان دهه هفتاد، یک راه حل ساده برای رفع بسیاری از مشکلات مربوط به جا به جای Pascal به ماشینهای مختلف، تولید کد میانی مستقل از ماشین برای یک ماشین مجازی انتزاعی بود (Barron، ۱۹۸۱). البته، این ماشین باید روی بسیاری از بسترها پیاده می شد، و به اینصورت برنامه های Pascal در هرجایی قابل اجرا می شدند . هرچند این ایده ساده برای چندین سال کاربرد گسترده ای پیدا کرد، هرگز واقعاً بعنوان یک راه حل عمومی برای مشکلات جابجایی در زبانهای دیگر، خصوصاً C، مطرح نشد. حدود ۲۵ سال بعد، ایده مهاجرت کد در سیستم های ناهمگن بوسیله زبانهای اسکریپتی<sup>۴۴۵</sup> و زبانهای دارای قابلیت جابجایی بالا<sup>۴۴۶</sup> از قبیل Java مورد استفاده قرار گرفت. بطور خلاصه، شیوه مورد استفاده در این راه حلها همان شیوه بکار رفته برای انتقال Pascal است . وجه مشترک تمامی راه حل های این گونه، استوار بودن براساس ماشین مجازی (فرآیندی) است که یا مستقیماً کد منبع (مانند زبانهای اسکریپتی) و یا به نوعی کد میانی تولید شده توسط یک کمپایلر (مانند Java) را اجرا می کند. بودن در مکان و زمان مناسب برای توسعه دهندها زبان های برنامه نویسی بسیار مهم است.

<sup>443</sup> heterogeneous systems

<sup>444</sup> homogeneous

<sup>445</sup> scripting languages

<sup>446</sup> highly portable languages

اخيراً اقداماتي برای کاهش ميزان وابستگی به زبانهای برنامه نويسي انجام شده است . به خصوص اينکه راه حلهاي برای مهاجرت فرآيندتها و بلکه برای مهاجرت کل محيط هاي محاسباتي پيشنهاد شده است. ايده اصلی در اين راه حل هاي جديد آن است که کل محيط را بخش بندی کرده و برای فرآيندها ديدگاه خودشان نسبت به محيط محاسباتي را ايجاد کند.

در صوريكه بخش بندی<sup>۴۴۷</sup> به طور مناسب انجام شود، می توان بخش را از سیستم زيرین تفکيك کرد و آن را واقعاً به ماشين ديگري مهاجرت داد. به اين ترتيب ، مهاجرت عملاً شکلي از جابجايی قوى برای فرآيندها خواهد شد، چون از آن پس می توان بخش را در هر زمان حين اجرا انتقال داده و پس از اتمام مهاجرت، درست از همان محل توقف مجدداً ازسر گرفت. بعلاوه ، می توان بسياري از پيچيدگي هاي مربوط به مهاجرت فرآيندها در رابطه با پيوند آنها به منابع محلی را حل نمود، چون اين پيوندها در بسياري از موارد براحتي حفظ می شوند. به اين معنی که ، منابع محلی غالباً بخشی از محيطی هستند که فرآيند به آن مهاجرت می کند.

دلail متعددی می توان برای مهاجرت تمام محيط مطرح کرد؛ اما شايد مهمترین آن چنین باشد که در صورتی که لازم باشد ماشيني به هر دليل از دور خارج شود، بازهم امكان ادامه عمليات وجود خواهد داشت . بعنوان مثال، فرض کنيد که در يك خوش سرور، مدیر اجرائي سیستم ها تصميم به خاموش کردن يا تعويض يكى از ماشينها بگيرد، اما نباید فرآيندهاي در حال اجرا را متوقف سازد. بلکه در عوض می تواند موقتاً محيطی را متوقف کرده ، آنرا به ماشين ديگر منتقل کند(در آنجا در کنار محيط هاي موجود ديگر قرار خواهد گرفت)، و مجدداً آنرا از حالت توقف خارج کند. واضح است که اين روش مديريت برای محيط هاي محاسباتي و فرآيندهاي آنها که قرار است در زمان هاي خيلي طولاني فعالیت کنند بسيار قدرتمند می باشد.

اجازه دهيد نمونه اي از مهاجرت ماشينهای مجازی را در نظر بگيريم که در مرجع (۲۰۰۵) Clark و گروه همكاران تشریح شده است. نويسندگان روی مهاجرت بي درنگ يك سیستم عامل مجازی شده<sup>۴۴۸</sup> تمرکز کرده اند که در خوشه اى از سرورها قابل مشاهده است و اتصال محكم<sup>۴۴۹</sup> آنها از طریق يك شبکه محلی مشترک ایجاد می شود. در اینصورت ، مهاجرت با دو مشکل عمدہ روپرو خواهد بود، يكى مهاجرت کل شکل حافظه و ديگري مهاجرت پيوندها به منابع محلی.

در مورد مشکل اول، در کل سه روش برای مهاجرت وجود دارد( که می توانند ترکیب شوند):

۱. هل دادن صفحات حافظه به ماشين جديد و ارسال مجدد صفحاتي که بعداً تغيير می یابند در حين فرآيند مهاجرت.
۲. متوقف سازی ماشين مجازی فعلی ؛ مهاجرت حافظه ، و آغاز ماشين مجازی جديد.

<sup>447</sup> compartmentalization

<sup>448</sup> Virtualized

<sup>449</sup> Tightly Coupled

۲. ایجاد امکان داخل کشیدن صفحات جدید در صورت نیاز توسط ماشین جدید ، یعنی ایجاد امکان آغاز فوری فرآیندها در یک ماشین مجازی جدید و کپی شدن صفحات حافظه در صورت وجود تقاضا .

چنانچه ماشین مجازی در حین مهاجرت در حال اجرای سرویس زنده ای<sup>۴۵۰</sup> - یا با بیان دیگر، سرویس پیوسته - باشد ، گزینه دوم ممکن است باعث زمان از کار افتادگی غیر قابل قبول سرویس شود. از طرف دیگر، اتخاذ شیوه بر پایه تقاضا که در گزینه سوم ارائه می شود، ممکن است باعث طولانی شدن بیش از حد زمان لازم برای مهاجرت شود. بعلاوه، از آنجایی که انتقال مجموعه کاری فرآیندهای مهاجرت کرده به ماشین جدید بسیار زمانبر است ، احتمالاً با افت کارآیی هم مواجه خواهیم شد . در روش پیشنهادی<sup>(۴۵۱)</sup> Clark و گروه همکاران از شیوه پیش کپی<sup>۴۵۱</sup> ، یعنی ترکیب گزینه اول با یک فاز توقف و کپی<sup>۴۵۲</sup> در گزینه دوم ، استفاده می شود . با چنین ترکیبی مدت از کار افتادگی سرویس<sup>۴۵۳</sup> ممکن است به ms ۲۰۰ یا حتی کمتر تقلیل یابد.

در مورد منابع محلی باید گفت چنانچه فقط با یک سرور خوشه ای سروکار داشته باشیم ، مسائل تا حدودی ساده تر خواهد شد. اولاً، از آنجاییکه فقط یک شبکه وجود دارد، فقط نیازمند اعلام پیوند آدرس شبکه-به- MAC جدید خواهیم بود تا مشتریان بتوانند بكمک آن با فرآیندهای مهاجرت کرده در واسط شبکه ای صحیح تماس برقرار کنند. و ثانیاً، اگر بتوان فرض کرد که منبع ذخیره ساز طبقه جداگانه ای را به خود اختصاص می دهد ( مطابق آنچه در شکل ۳-۱۲ مشاهده می کنید)، مهاجرت پیوند به فایلها هم ساده تر خواهد شد.

نتیجه و تأثیر نهایی این است که، بجای مهاجرت فرآیندها، عملاً شاهد جابجایی کل سیستم عامل در بین ماشینهای مختلف خواهیم بود .

### ۶-۳- خلاصه

از آنجایی که فرآیندها مبنای ارتباط بین ماشینهای مختلف را فراهم می کنند، بنابراین نقش اساسی در سیستم های توزیعی بر عهده دارند. یکی از مهمترین مسائل در ارتباط با فرآیندها، نحوه سازماندهی درونی آنها و بطور اخص، پشتیبانی یا عدم پشتیبانی از چند نخ کنترل است. کاربرد ویژه نخها در سیستم های توزیعی موجب ادامه استفاده از پردازنده در حین انجام عملیات I/O مسدود شونده است. به این ترتیب، امکان ساخت سرورهای پربازدهی فراهم خواهد شد که چندین نخ را به صورت موازی با هم اجرا کرده و برخی از آنها ممکن است تا زمان تکمیل I/O دیسک یا ارتباط شبکه ای منتظر و مسدود باقی بمانند.

<sup>450</sup> live service

<sup>451</sup> pre-copy

<sup>452</sup> stop-and-copy phase

<sup>453</sup> service downtimes

سازماندهی برنامه کاربردی توزیعی به صورت مشتریها و سرورها هم بسیار مفید می باشد . فرآیندهای مشتری غالباً واسطه های کاربر را پیاده سازی می کنند که ممکن است از صفحه نمایشگاهی بسیار ساده تا واسطه های پیشرفته متفاوت باشند و می توانند روی اسناد ترکیبی کار کنند . از این گذشته، یکی از اهداف نرم افزار مشتری کسب شفافیت توزیعی از طریق مخفی سازی جزئیات مربوط به ارتباط با سرورها ، محل استقرار فعلی این سرورها و چند نسخه ای یا یک نسخه ای بودن آنهاست. بعلاوه، نرم افزار مشتری تا حدودی مسئول مخفی سازی خرابی ها و بازسازی خرابی هاست.

گرچه سرورها غالباً پیچیده تر از مشتری ها هستند، اما مسائل طراحی محدودی دارند. به عنوان مثال، سرورها ممکن است تکراری یا همرونده بوده، یک یا چند سرویس را پیاده سازی کرده و بدون حالت یا دارای حالت باشند. مسائل طراحی دیگر مربوط به آدرس دهی سرویسها و مکانیزم ها جهت توقف سرور پس از صدور درخواست سرویس بوده که احتمالاً در حال پردازش می باشد.

بایستی در سازماندهی سرورها به صورت خوش توجه زیادی صرف کرد. یکی از اهداف مشترک در این خوش ها، مخفی سازی سازمان دهی داخلی خوش از دید دنیای خارج است. به این معنا که سازمان خوش مخفی در برابر برنامه های کاربردی است. بدین منظور، اغلب خوش ها فقط از یک نقطه ورودی برای تقدیم پیامها به سرورهای داخل خوش استفاده می کنند. یکی از مشکلات چالشی در طراحی خوش ها تعویض شفاف این نقطه ورودی بوسیله راه حل های توزیعی می باشد.

یکی از موضوعات مهم در سیستم های توزیعی ، مهاجرت کد بین ماشینهای مختلف است. دو دلیل عمده برای پشتیبانی از مهاجرت کد، افزایش کارآیی و انعطاف پذیری است. در مواردیکه ارتباطات هزینه بر باشد، می توان با انتقال برخی از محاسبات از سرور به مشتری و فراهم کردن امکان انجام حداکثر پردازش محلی ممکنه برای مشتری اقدام به کاهش ارتباطات نمود. چنانچه مشتری بتواند به صورت پویا نرم افزار لازم برای ارتباط با یک سرور خاص را دانلود کند، انعطاف پذیری افزایش خواهد یافت. نرم افزار دانلود شده می تواند مخصوص آن سرور باشد، بدون اینکه نیازی باشد که مشتری از قبل آن را نصب کرده باشد.

مهاجرت کد مشکلاتی در واسطه با استفاده از منابع محلی بوجود می آورد . برای حل این مشکلات یا باید منابع را هم مهاجرت داد، یا پیوندهای جدید به منابع محلی ماشین هدف ایجاد شود، و یا از مراجع شبکه ای جهانی استفاده نمود . مشکل دیگر اینکه در مهاجرت کد باید مسئله ناهمگن بودن مورد توجه قرار گیرد. در وضعیت فعلی ، بهترین راه حل در این واسطه استفاده از ماشینهای مجازی است. این ماشین ها ممکن است یا به صورت ماشینهای مجازی فرآیندی باشند که مثلاً در Java بچشم می خورد و یا از ناظرهای ماشین مجازی استفاده نمود که امکان مهاجرت مؤثر مجموعه ای از فرآیندها به همراه سیستم عامل زیرین آنها را فراهم می آورد.

## مسائل فصل

- ۱- در این مسئله باید خواندن فایل با استفاده از سرور فایل تک نخی را با استفاده از سرور چند نخی مقایسه کنید. ۱۵ msec طول می کشد تا یک درخواست را به کار گرفت، آنرا توزیع کرده و پردازش لازم را انجام داد، البته با این فرض که داده های لازم برای اینکار در حافظه پنهان در حافظه اصلی قرار داشته باشند. در صورت نیاز به عملیات دیسک، که در یک سوم موارد هم لازم است، ۷۵ msec دیگر هم برای خوابیدن نخ به رقم بالا اضافه کنید. در صورت تک نخی بودن، سرور قادر به انجام چه تعداد درخواست در واحد ثانیه خواهد بود؟ در صورت چند نخی بودن چطور؟
- ۲- آیا محدود کردن تعداد نخها در فرآیند سرور کار معقولانه ای است؟
- ۳- در متن درس، نمونه ای از سرورهای فایل چند نخی را تشریح و دلایل برتری آن در مقایسه با سرور تک نخی شده و سرور ماشین حالت محدود را ذکر کردیم. آیا مواردی وجود دارد که سرورهای تک نخی بهتر باشند. یک مثال ذکر کنید.
- ۴- مرتبط ساختن ایستای فقط یک نخ با یک فرآیند سبک وزن ایده چندان مناسبی نیست. چرا؟
- ۵- داشتن فقط یک فرآیند سبک وزن در هر فرآیند هم چندان ایده مناسبی نیست. چرا؟
- ۶- طرح ساده ای را شرح دهید که در آن تعداد فرآیندهای سبک وزن موجود به اندازه تعداد نخهای قابل اجرا باشد.
- ۷- X، پایانه کاربر را بعنوان میزبان سرور تعیین می کند، در حالیکه برنامه کاربردی تحت عنوان مشتری خوانده می شود. آیا اینکار صحیح است؟
- ۸- پروتکل X دچار مشکلات مقیاس پذیری است. چه راه حلی برای این مشکل ارائه می دهید؟
- ۹- همانطور که در متن درس هم گفتیم، نرم افزار های پنهان<sup>۴۵۴</sup> می توانند با فراخوانی هر یک از نسخه ها از شفافیت چند نسخه ای پشتیبانی کنند. آیا (طرف سرور) یک برنامه کاربردی هم می تواند در معرض فراخوانی های کپی شده قرار گیرد؟
- ۱۰- ساخت یک سرور همرونده از طریق تولید یک فرآیند در مقایسه با سرورهای چند نخی، مزايا و معایبي دارد. چند نمونه از هردو را ذکر کنید.
- ۱۱- طرحی از یک سرور چند نخی ترسیم کنید که با استفاده از سوکتها بعنوان واسطه لایه انتقال با سیستم عامل از پروتکل های متعددی پشتیبانی می کند.
- ۱۲- چگونه می توان از دور زدن مدیر پنجره به توسط برنامه های کاربردی و ایجاد بهم ریختگی در صفحه جلوگیری کرد؟
- ۱۳- سروری که ارتباط TCP/IP با یک مشتری دارد بدون حالت یا دارای حالت است؟
- ۱۴- یک سرور وب را در نظر بگیرید که شامل جدولی است که در آن آدرسهای IP مشتری به آخرین صفحات وبی که دسترسی پیدا کرده، نگاشته شده اند. وقتی مشتری به سرور متصل می شود، سرور در

<sup>454</sup> Proxies

جدول خود به دنبال مشتری می‌گردد و در صورت یافتن شدن، صفحه ثبت شده را باز می‌گرداند.  
چنین سروی دارای حالت است یا بدون حالت؟

۱۵- جابجایی قوی در سیستم‌های UNIX از طریق ایجاد امکان منشعب ساختن فرزند واقع روی یک ماشین دور قابل پشتیبانی است. نحوه کار را توضیح دهید.

۱۶- از شکل ۳-۱۸ می‌توان دریافت که جابجایی قوی با اجرای کد مهاجرت کرده در فرآیند هدف سازگار نیست. یک نمونه عکس ذکر کنید.

۱۷- فرض کنید فرآیند  $P$  نیازمند دسترسی به فایل  $F$  باشد که به صورت محلی روی ماشینی قابل دسترسی است که هم اکنون  $P$  ببروی آن در حال اجراست. در صورتیکه  $P$  به ماشین دیگری انتقال پیدا کند،  $P$  بازهم نیازمند دسترسی به فایل  $F$  خواهد بود. اگر پیوند فایل به ماشین ثابت باشد، ارجاع تمام سیستمی<sup>۴۵۵</sup> به  $F$  چگونه قابل اجراست؟

۱۸- به تفصیل شرح دهید که در حالت تقدیم<sup>۴۵۶</sup> TCP، بسته‌های TCP به همراه اطلاعات مربوط به آدرس‌های منبع و مقصد در سرآیندهای مختلف چگونه جریان پیدا می‌کنند؟

---

<sup>455</sup> System-wide

<sup>456</sup> Handoff

## ارتباطات

ارتباطات میان فرآیندی قلب تپنده تمامی سیستم های توزیعی محسوب می شود. در واقع، مطالعه سیستم های توزیعی بدون بررسی دقیق روش‌های تبادل اطلاعات از طریق فرآیندهای ماشینهای مختلف امری بیهوده است. در سیستم های توزیعی، ارتباطات همواره بر اساس ردکردن پیام سطح پایین ارائه شده از شبکه زیرین (زیربنایی<sup>۴۵۷</sup>؟؟) استوار است. ارتباطات سریع از طریق ردکردن پیام مشکلتر از به کارگیری عملیات اولیه<sup>۴۵۸</sup> حافظه اشتراکی<sup>۴۵۹</sup> است که ویژگی روش‌های (بسترهاي<sup>۴۶۰</sup>؟؟؟؟) غیرتوزیعی محسوب می شود. سیستم های توزیعی مدرن غالباً متشکل از هزاران یا حتی میلیونها فرآیندی است که در سرتاسر شبکه های دارای ارتباطات نامطمئن از جمله اینترنت پراکنده شده اند. تا زمانیکه امکانات ارتباطی ابتدایی شبکه های کامپیوتري تعویض و یا جایگزین نشوند، توسعه برنامه های کاربردی توزیعی پردازمنه تقریباً غیرممکن خواهد بود. در این فصل، ابتدا راجع به قوانینی به نام پروتکل بحث می کنیم که فرآیندهای ارتباطی باستی براساس آن صورت گرفته و روی لایه ای کردن این پروتکل ها تمرکز خواهیم کرد. سپس نظری به سه مدل ارتباطی پرکاربرد، شامل فراخوانی روال از راه دور<sup>۴۶۱</sup> (PRC)، میان افزار پیام گرا<sup>۴۶۰</sup> (MOM) و رشته ای کردن داده ها<sup>۴۶۲</sup> خواهیم داشت. همچنین درمورد مشکل عمومی ارسال داده برای چندین گیرنده، یا چندپخشی<sup>۴۶۳</sup>، بحث خواهیم کرد.

بعنوان اولین مدل ارتباطات در سیستم های توزیعی، فراخوانی روال از راه دور (RPC) را در نظر می گیریم. هدف از RPC مخفی سازی بخش عظیمی از ظرافتهاي ردکردن پیام است و به همین دلیل، برای برنامه های کاربردی مشتری- سرور ایده آل می باشد.

در بسیاری از برنامه های کاربردی توزیعی، ارتباطات از الگوی بسیار خشک تبادل مشتری- سرور<sup>۴۶۴</sup> تبعیت نمی کند. در نتیجه، در چنین مواردی بررسی خود پیام مفیدتر خواهد بود. با این وجود، امکانات ارتباطات سطح پایین شبکه های کامپیوتري، به دلیل عدم شفافیت توزیعی، از بسیاری جهات نامناسب است. یک راه حل، استفاده از مدل صف‌بندی پیام سطح بالا<sup>۴۶۵</sup> است که ارتباطات آن تاحد زیادی مشابه سیستم های پست الکترونیک صورت می پذیرد. اهمیت موضوعی میان افزار پیام گرا (MOM) به حدی است که می توان یک مبحث کامل را به آن اختصاص داد.

با ظهور سیستم های توزیعی چندرسانه ای مشخص شد که بسیاری از سیستم ها از ارتباطات رسانه های پیوسته ای از قبیل رسانه های صوتی و شکلی پشتیبانی نمی کنند. در واقع لازم است با ایجاد

<sup>457</sup> primitive

<sup>458</sup> shared memory

<sup>459</sup> Remote Procedure Calls

<sup>460</sup> Message-Oriented Middleware

<sup>461</sup> data streaming

<sup>462</sup> multicasting

<sup>463</sup> client-server

<sup>464</sup> high-level message-queuing model

رشته، از جریان پیوسته پیامهای دچار محدودیت زمان بندی مختلف پشتیبانی شود. در مورد جریانات در بخش جدأگانه بحث خواهیم کرد.

در پایان، از آنجاییکه دانش ما نسبت به ایجاد امکانات چندپخشی افزایش یافته، راه حلهای جدید و بسیار مناسبی هم جهت انتشار داده ها بوجود آمده است. از اینرو، بخش پایانی فصل را به همین موضوع اختصاص داده ایم.

#### ۴-۱- مبانی

پیش از شروع بحث در مورد ارتباطات در سیستم های توزیعی، ابتدا برخی از مسائل ریشه ای در زمینه ارتباطات را به طور اجمالی مورد بحث قرار خواهیم داد. در بخش بعد، مختصرآ در مورد پروتکلهای ارتباطات شبکه ای، از جمله آنها یکی که اساس سیستم های توزیعی را تشکیل می دهد، بحث خواهیم کرد. سپس، با دسته بندی انواع مختلف ارتباطات در سیستم های توزیعی، روش متفاوتی را معرفی می کنیم.

#### ۴-۱-۱- پروتکلهای لایه ای

در سیستم های توزیعی، به دلیل عدم وجود حافظه اشتراکی، تمامی ارتباطات براساس ارسال و دریافت پیامها (ی سطح پایین) استوار است. در صورتیکه فرآیند *A* بخواهد با فرآیند *B* ارتباط برقرار کند، ابتدا در فضای آدرس خود یک پیام ایجاد می کند. سپس با اجرای فرآخوان سیستمی<sup>۴۶۵</sup>، به سیستم عامل امکان می دهد تا پیام را از طریق شبکه به *B* ارسال کند. هر چند این ایده بسیار ساده می نماید، *A* و *B* بایستی در مورد معنای بیتهاي ارسال شده به توافق برسند، وگرنه احتمال بروز هر نوع بحرانی وجود خواهد داشت. چنانچه *A* داستان کوتاه تازه ای را به زبان فرانسه ارسال کند که براساس کد حروف IBM EBCDI دریافت لیست موجودی کالای سوپرمارکت نوشته شده به زبان انگلیسي و کدگذاری شده در ASCII را داشته باشد، آنگاه ارتباط در حد بهینه نخواهد بود.

لازم است راجع به موارد مختلف توافق شود، از قبیل اینکه برای ارسال بیت صفر از چه مقدار ولت و برای بیت یک از چه مقدار بایستی استفاده شود؟ گیرنده بیت آخر را چگونه شناسایی می کند؟ یا درصورت آسیب دیدگی یا مفقود شدن، پیام چگونه قابل ردیابی خواهد بود یا برای یافتن پیام به چه صورت عمل خواهد شد. اعداد، جریانات و دیگر اقلام داده ای چه طولی داشته و چگونه نمایش داده می شوند؟ به طور خلاصه، بایستی در تمامی سطوح و موارد، از جزئیات سطح پایین انتقال بیت گرفته تا جزئیات سطح بالای نحوه بیان اطلاعات توافق صورت گیرد.

به منظور تسهیل بحث در مورد سطوح و مسائل متعدد مربوط به ارتباطات، مؤسسه بین المللی استاندارد<sup>۴۶۶</sup> (ISO) با ارائه مدل مرجع، اقدام به شناسایی دقیق سطوح مختلف مربوطه نمود. بعلاوه، اسامی استانداردی را برای هریک تعیین و مسئولیت هر سطح را مشخص نموده است. این مدل، **مدل مرجع میان اتصالی سیستم های**

<sup>465</sup> system call

<sup>466</sup> International Standard Organization

**باز<sup>۴۶۷</sup>- OSI** (Day and Zimmerman، ۱۹۸۳) یا به طور خلاصه ISO OSI نامیده می شود. لازم به ذکر است که پروتکل های ایجادی بعنوان بخشی از مدل OSI هرگز مورد کاربرد وسیع و گسترده قرار نگرفت و امروزه دیگر به طور کامل منسخ شده اند. با این وجود، بررسی این مدل در درک شبکه های کامپیوتري بسیار مفید خواهد بود. هرچند قصد نداریم در اینجا شرح کاملی از مدل مذکور و اثرات جانبی آن ارائه دهیم ، تعریف مختصر آن خالی از لطف نخواهد بود. برای مطالعه دقیق تر موضوع به Tanenbaum (۲۰۰۳)

مراجعه شود.

مدل OSI جهت ایجاد ارتباط بین سیستم های باز طراحی شده است. سیستم باز سیستمی است که آمادگی دارد تا براساس قوانین استاندارد مربوط به قالب، محتوا و معنای پیام های ارسال و دریافت شده ، با هر سیستم باز دیگری ارتباط برقرار نماید. این قوانین تحت عنوان **پروتکل<sup>۴۶۸</sup>** خوانده و مدون می شوند . برای آنکه یک گروه کامپیوتور بتوانند در شبکه با هم ارتباط برقرار نمایند، بایستی همگی از پروتکل های مشترکی استفاده کنند. در این رابطه ، دو نوع **پروتکل<sup>۴۶۹</sup>** با برخی تفاوتها وجود دارد. در **پروتکل های اتصال گرا<sup>۴۷۰</sup>**، فرستنده و گیرنده پیش از تبادل داده ها ، ابتدا به وضوح یک اتصال ایجاد کرده و احتمالاً در مورد پروتکل مورد استفاده خود بحث و چانه زنی می نمایند . در پایان، اتصال بایستی آزاد (متوقف) شود. دستگاه تلفن نمونه ای از سیستم های اتصال گرا است. اما در **پروتکل های بدون اتصال<sup>۴۷۰</sup>**، نیازی به ایجاد هیچ تنظیم قبلی وجود ندارد. اولین پیام به محف آماده شدن توسط فرستنده ارسال می شود . انداختن نامه در صندوق پست نمونه ای از ارتباطات بدون اتصال است. در کامپیوتراها، هردو صورت ارتباطات اتصال گرا و بدون اتصال کاربرد دارد.

در مدل OSI، مطابق شکل ۴-۱ ، ارتباط به هفت سطح یا لایه تقسیم می شود و هر لایه مسئول جنبه خاصی از ارتباط می باشد. به این ترتیب، مسئله را می توان به چندین بخش معنادار تقسیم و هریک را مستقل از بخش های دیگر حل نمود. بعلاوه ، هر لایه رابطی<sup>۴۷۱</sup> برای لایه فوقانی خود ایجاد می کند. این رابط مجموعه ای از عملیاتی است که رویهم رفته معرف سرویس قابل ارائه توسط لایه به کاربران خود ارائه خواهد بود.

شکل ۴-۱ لایه ها، رابط ها و پروتکلهای مدل OSI (داخل شکل: برنامه کاربردی، پروتکل برنامه کاربردی، ارائه<sup>۴۷۲</sup>، پروتکل ارائه، جلسه<sup>۴۷۳</sup>، پروتکل جلسه، انتقال<sup>۴۷۴</sup>، پروتکل انتقال، شبکه<sup>۴۷۵</sup>، پروتکل شبکه، پیوند داده<sup>۴۷۶</sup>، پروتکل پیوند داده، فیزیکی<sup>۴۷۷</sup>، پروتکل فیزیکی).

<sup>467</sup> Open Systems Interconnection Reference Model

<sup>468</sup> protocol

<sup>469</sup> connection-oriented protocols

<sup>470</sup> connectionless protocols

<sup>471</sup> interface

<sup>472</sup> presentation

<sup>473</sup> session

<sup>474</sup> transport

<sup>475</sup> network

<sup>476</sup> data link

هنگامیکه فرآیند A روی ماشین ۱ قصد برقراری ارتباط با فرآیند B روی ماشین ۲ را داشته باشد، پس از ساخت پیام، آنرا به لایه برنامه کاربردی ماشین خود انتقال می‌دهد. این لایه ممکن است بعنوان مثال یک روال کتابخانه ای باشد، ولی به روش دیگری نیز قابل پیاده سازی است (مثلًا داخل سیستم عامل یا روی یک پردازشگر شبکه بیرونی و غیره). سپس نرم افزار لایه کاربردی، یک سرپیام<sup>۴۷۸</sup> را به بخش اولیه پیام اضافه نموده و پیام را نهایتاً از طریق رابط لایه ۶/۷ به لایه ارائه انتقال می‌دهد. لایه ارائه هم متقابلاً سرپیام خود را اضافه و نتیجه را به لایه جلسه ارسال می‌کند و الى آخر. برخی لایه‌ها علاوه بر افزودن سرپیام، پشت بندی<sup>۴۷۹</sup> را هم به انتهای آن اضافه می‌کنند. در پایان، لایه فیزیکی با قرار دادن پیام روی رسانه ارسال فیزیکی، واقعًا پیام را که ممکن است در این حالت مشابه شکل ۴-۲ باشد، ارسال می‌کند.

شکل ۴-۲-نمونه ای از شکل پیام در شبکه (داخل شکل: سرپیام لایه انتقال، سرپیام لایه جلسه، سرپیام لایه ارائه، سرپیام لایه برنامه کاربردی).

پیام پس از رسیدن به ماشین<sup>۲</sup>، به سمت بالا می‌رود، در هر لایه سرپیام مربوطه جدا شده و بازبینی می‌شود. در نهایت، پیام به گیرنده یعنی به فرآیند ۲ می‌رسد و این فرآیند در مسیر عکس به آن جواب می‌دهد. از اطلاعات داخل سرپیام لایه <sup>n</sup> جهت پروتکل لایه <sup>n</sup> هم استفاده می‌شود.

برای درک صحیح اهمیت پروتکل‌های لایه ای شده، ارتباطات بین دو شرکت خطوط هوایی Zippy و تهیه غذای Mushy Meals را در نظر بگیرید. هر ماه، رئیس خدمات مسافربری Zippy از طریق منشی خود با مدیر فروش Mushy تماس گرفته و تعداد ۱۰۰۰۰ بسته جوجه شفارش می‌دهد. سابقاً کار سفارش دهی از طریق پست انجام می‌شد. اما به دلیل نامناسب بودن خدمات پستی، منشی ایندو شرکت تصمیم می‌گیرند که بوسیله پست الکترونیک باهم ارتباط برقرار کنند. چون در پروتکلهای مورد توافق آنها در مورد انتقال فیزیکی سفارشات بحث شده نه محتواي آنها، منشی ایندو شرکت می‌توانند بدون ایجاد کوچکترین مزاحمت یا حتی اطلاع رؤسای خود، کار سفارشات را انجام دهند.

به همین ترتیب، رئیس خدمات مسافربری می‌تواند به راحتی سفارش جوجه را کنسل کرده و به جای آن، غذای ویژه Mushy را سفارش دهد. توجه کنید که این تصمیم جدید هیچ ارتباطی با منشی‌ها ندارد و ممکن است آنها حتی کوچکترین اطلاعی هم از سفارش جدید نداشته باشند. مسئله مهم آن است که در این مورد، دو لایه داریم یکی رؤسا و دیگری منشی‌ها. به علاوه، هر لایه پروتکل خاصی دارد که مستقل از دیگری قابل تغییر است. دقیقاً همین استقلال مبحث پروتکلهای لایه ای را به موضوعی جذاب تبدیل کرده است. در واقع هر لایه را می‌توان بدون کوچکترین اثری بر لایه دیگر و بر اساس آخرین فن آوری روز اصلاح نمود.

<sup>477</sup> physical

<sup>478</sup> header

<sup>479</sup> trailer

در مدل OSI فقط دو لایه وجود ندارد ، بلکه مطابق شکل ۴-۱ ، با هفت لایه سروکار داریم. مجموعه پروتکل های مورد استفاده در یک سیستم خاص، **مجموعه پروتکل**<sup>۴۸۰</sup> یا **بسته پروتکل**<sup>۴۸۱</sup> نامیده می شود. مسأله مهم دیگر، تمایز قائل شدن بین مدل مرجع و پروتکلهای OSI واقعی آن است. همانطور که قبل ام اشاره کردیم، پروتکل های OSI هرگز کاربرد گسترده نیافتدند. در حالی که پروتکل های ساخته شده برای اینترنت از قبیل TCP و IP رواج فراوانی دارند. در بخش های بعد، هر یک از لایه های OSI را به نوبت و مختصرآ (از پایین به بالا) مورد بررسی قرار خواهیم داد. همچنین، به جای ارائه نمونه هایی در مورد پروتکل های OSI ، هر کجا مناسب باشد به برخی از پروتکل های اینترنتی مربوط به هر لایه اشاره خواهیم کرد.

## پروتکل های سطح پایین تر

بحث را با سه لایه زیرین مجموعه پروتکل OSI آغاز می کنیم. این لایه ها در مجموع، کارکردهای اساسی یک شبکه کامپیوتري را پیاده سازی می کنند.

لایه فیزیکی مسئول ارسال و ۰۱ است. مقدار ولت لازم برای ارسال ۰۰۰ ، تعداد بیتهاي قابل ارسال در هر ثانیه و امكان ارسال همزمان در هر دو جهت از جمله مسائل کلیدی در لایه فیزیکی هستند. بعلاوه، ابعاد و شکل اتصال دهنده شبکه (plug) و همچنین پین ها و معنی هریک مورد بحث قرار می گیرد.

پروتکل لایه فیزیکی به استاندارد سازی رابط های الکتریکی ، مکانیکی و سیگنال دهی می پردازد تا هر زمانی که ماشین بیت صفر ارسال می کند، واقعاً هم به صورت بیت صفر دریافت شود نه بیت یک . استاندارد های لایه فیزیکی متعددی (برای رسانه های مختلف) ایجاد و ارائه شده که از آن جمله می توان به استاندارد RS-232-C

برای خطوط ارتباطی سریال اشاره کرد.

لایه فیزیکی صرفاً مسئول ارسال بیت است. مادامیکه هیچ خطایی انجام نشود، همه چیز روال عادی خود را طی خواهد کرد. اما ، شبکه های ارتباطی واقعی در معرض خطا قرار دارند. به همین دلیل، برای ردیابی و اصلاح این دست خطاها نیازمند مکانیزم های لازم هستیم. در واقع، این مکانیزم ها وظیفه اصلی لایه پیوند داده است. این لایه بیت ها را به چندین واحد که در برخی موارد قاب<sup>۴۸۲</sup> نامیده می شود، دسته بندي کرده و سپس بر دریافت درست هر قاب نظارت می کند.

وظیفه لایه پیوند داده، قرار دادن یک الگوی خاص بیت در انتهای ابتدای هر فریم جهت نشانه گذاری آنهاست. همچنین با جمع کردن تمامی بیت های قاب به روشنی خاص، اقدام به محاسبه **حاصلجمع آزمایشی**<sup>۴۸۳</sup> می کند. لایه پیوند داده این حاصلجمع آزمایشی را به قاب ضمیمه می کند. گیرنده پس از دریافت قاب، حاصلجمع آزمایشی را بر حسب داده دریافت شده محاسبه نموده و نتیجه را با حاصلجمع آزمایشی مربوط به قاب مقایسه می کند. در صورت تطابق، قالب

<sup>480</sup> protocol suite

<sup>481</sup> protocol stack

<sup>482</sup> frame

<sup>483</sup> checksum

صحیح بوده و پذیرفته می شود. در غیراینصورت، گیرنده از فرستنده درخواست ارسال مجدد آنرا می نماید. (در بخش سرپیام) به هریک از قالب ها یک عدد ترتیب اختصاص می یابد تا بتوان هریک را به دقت شناسایی نمود.

در شبکه های محلی (LAN)، معمولاً نیازی به مکان یابی گیرنده توسط فرستنده وجود ندارد، بلکه صرفاً پیام توسط فرستنده روی شبکه قرار گرفته و گیرنده آنرا اخذ می کند. ولی در شبکه های گسترده، تعداد زیادی ماشین وجود دارد که هر یک با تعدادی خط ارتباطی به ماشین های دیگر وصل می شود و از این نظر بسیار شبیه نقشه بزرگی از شهرها و راههای موافقاتی بین آنهاست. برای آنکه یک پیام از فرستنده به گیرنده برسد، ممکن است چند گذر<sup>۴۸۴</sup> داشته و در هر گذر از یک خط استفاده کند. مسئله نحوه انتخاب بهترین مسیر، مسیریابی<sup>۴۸۵</sup> نامیده شده واساساً وظیفه اصلی لایه شبکه است.

اما این واقعیت که کوتاه ترین مسیر همیشه بهترین نیست، تا حدودی باعث پیچیدگی مسئله مسیریابی می شود. آنچه در واقع عامل این مشکل می باشد، مقدار تأخیرها در یک مسیر مفروض است که آنهم به نوبه خود، با میزان ترافیک و تعداد پیام های منتظر برای ارسال روی خطوط مختلف ارتباط دارد. بنابراین، تأخیر می تواند در طول زمان تغییر کند. در برخی الگوریتم های مسیریابی سعی شده مسئله تطبیق با تغییر بار لحاظ شود؛ در حالیکه در برخی دیگر، تصمیمات فقط بر اساس میانگین های بلند مدت گرفته می شود.

در حال حاضر، پرکاربردترین پروتکل شبکه، پروتکل بدون اتصال IP (پروتکل اینترنت<sup>۴۸۶</sup>) است که بخش مهمی از مجموعه پروتکل اینترنت محسوب می شود. **بسته IP**<sup>۴۸۷</sup> (اصطلاح فنی مورد استفاده برای پیام در لایه شبکه) را می توان بدون هیچ تنظیم و تغییری ارسال نمود. هریک از بسته های IP مستقل از دیگری به سمت مقصد مسیردهی می شود. بعلاوه، هیچ مسیر داخلی از پیش تعیین و انتخاب نمی شود.

### پروتکل های انتقال

لایه انتقال بخش آخر چیزی را تشکیل می دهد که مجموعه اساسی پروتکل شبکه نامیده می شود. دلیل نامگذاری آن است که لایه مذکور قادر به انجام کلیه سرویس هایی است که گرچه منطقاً لازمه ساخت برنامه های کاربردی شبکه می باشند، اما در رابط لایه شبکه ارائه نمی شود. به بیان دیگر، لایه انتقال، شبکه زیرین را برای سازنده برنامه کاربردی قابل استفاده می نماید. ممکن است برخی بسته ها در طی مسیر خود از فرستنده به گیرنده مفقود شوند. هر چند برخی برنامه های کاربردی قادر به بازسازی و اصلاح خطای خود هستند، برخی دیگر به دنبال ایجاد یک اتصال قابل اطمینان و بدون عیب هستند. ارائه این سرویس وظیفه لایه انتقال است. به این معنی که لایه انتقال بایستی قادر به تحويل

<sup>484</sup> hop

<sup>485</sup> routing

<sup>486</sup> Internet Protocol

<sup>487</sup> IP packet

پیام به لایه انتقال باشد و این تحویل بایستی بدون هیچ خدشه ای به پیام انجام شود.

لایه انتقال به محض دریافت پیام از لایه برنامه کاربردی آنرا به قطعات کوچکی تبدیل می‌کند. اندازه قطعات باید به حدی باشد که ارسال آنها امکان‌پذیر باشد. سپس به هر قطعه یک شماره ترتیب الصاق و ارسال می‌شود. بحث در مورد سرپیام لایه انتقال روی موضوعاتی از این قبیل تمرکز می‌کند: کدام بسته‌ها ارسال شده‌اند، کدام بسته‌ها دریافت شده‌اند، گیرنده چه مقدار فضای برای دریافت پیام در اختیار دارد، کدام پیام باید مجدد ارسال شود و سایر مواردی از این دست.

اتصالات انتقال مطمئن را (که بر حسب تعریف اتصال گرا هستند) می‌توان توسط خدمات شبکه اتصال گرا یا بدون اتصال با موفقیت کنترل نمود. در حالت اول، تمامی بسته‌ها به ترتیب صحیح خواهند رسید (البته اگر به مقصد برسند)، اما در حالت دوم ممکن است یکی از بسته‌ها مسیر دیگری را پیش‌گرفته و زودتر از بسته‌ای که پیش از آن ارسال شده، به مقصد برسد. این وظیفه نرم افزار لایه انتقال است که تمامی بسته‌ها را به صورتی بازگرداند که گویا اتصال انتقال شبیه یک لوله بزرگ عمل می‌کند – پیام در این لوله قرار گرفته و سالم و بدون کوچکترین آسیبی، به همان ترتیب ورود، خارج می‌شود. ایجاد این رفتار ارتباطی مرتب و منظم یکی از جنبه‌های مهم لایه انتقال است.

پروتکل انتقال اینترنت به نام **TCP** (پروتکل کنترل ارسال<sup>۴۸۸</sup>) خوانده شده و به تفصیل در کتاب **Comer (۲۰۰۶)** شرح داده شده است. امروزه از ترکیب **TCP/IP** بعنوان یک استاندارد کاربردی در ارتباطات شبکه ای استفاده می‌شود. بعلاوه، مجموعه پروتکل اینترنت از یک پروتکل انتقال بدون اتصال به نام **UDP** (پروتکل جهانی دیتاگرم<sup>۴۸۹</sup>) پشتیبانی می‌کند که در واقع همان IP قدیمی بعلاوه برخی اضافات جزئی است. در برنامه‌های کاربرانی که نیازی به پروتکل اتصال گراندارند، غالباً از **UDP** استفاده می‌شود. پروتکل‌های انتقالی جدید دیگری هم مرتبأ به بازار عرضه می‌شوند. بعنوان مثال، جهت پشتیبانی از انتقال داده بی درنگ<sup>۴۹۰</sup>، پروتکل انتقال بی درنگ<sup>۴۹۱</sup> (**RTP**) تعریف و ارائه شده است. **RTP** پروتکل چارچوبی بوده و قالب بسته‌ها را جهت داده‌های بی درنگ تعیین و تعریف می‌کند. اینکار بدون ایجاد مکانیزم‌های عملی تضمین تحویل پیام صورت می‌پذیرد. بعلاوه، پروتکلی جهت نظارت و کنترل بر انتقال داده‌های بسته‌ها **RTP** تعیین می‌کند (Schulzrinne، ۲۰۰۳).

### پروتکل‌های سطح بالاتر

OSI سه لایه دیگر هم در بالای لایه انتقال مشخص نموده، اما بطور عملی تاکنون فقط لایه برنامه کاربردی استفاده شده است. در واقع، در مجموعه پروتکل اینترنت، هر چیزی که بالاتر از لایه

<sup>488</sup> Transmission Control Protocol

<sup>489</sup> Universal Datagram Protocol

<sup>490</sup> real-time

<sup>491</sup> Real-time Transport Protocol

انتقال قرار گیرد در این گروه دسته بندي می شود. در اين بخش مشاهده خواهيم کرد که برای سистем هاي ميان افزاری، هيچيک از دو روش OSI و اينترنت واقعاً مناسب نیستند.

لایه جلسه<sup>۴۹۲</sup> در اصل نسخه تقویت شده لایه انتقال است. این لایه گفتگو را کنترل می کند تا بدین وسیله امكان پی گيري طرف گفتگو کننده فراهم آيد. بعلاوه، امكانات همگام سازی را ایجاد می کند. این امكانات به کاربر اجازه می دهد تا در انتقالات طولاني، نقاط آزمایشي<sup>۴۹۳</sup> را درج نماید تا در صورت بروز خرابي به جاي تکرار مجدد مسیر، فقط آخرین نقطه آزمایشي بررسی شود. در عمل، برنامه هاي کاربردي بسيار محدودي به لایه جلسه علاقمند بوده و به ندرت پشتيبانی می شوند. اين لایه حتی در مجموعه پروتوكل اينترنت هم وجود ندارد. اين در حالی است که مفهوم جلسه و پروتوكلهای مربوط به آن در روند ارائه راه حل برای ميان افزارها و خصوصاً در هنگام تعریف پروتوكلهای ارتباطات سطح بالاتر نقش مهمی ايفا می کند.

لایه ارائه، برخلاف لایه هاي زيرين خودکه مسئول دریافت بيت ها و ارسال موثق و مؤثر آنها به گيرنده هستند، مسئول معنای بيتهاست. بطوریکه اغلب پیامها متشکل از زنجيره هاي بيت تصادفي نیستند، بلکه حاوي اطلاعات سازماندهی شده اي از قبيل اسامي افراد، آدرس ها و مقدار موجودی آنها و غيره می باشد. در لایه ارائه، امكان تعریف گزارشات حاوي موضوعاتی از اين دست وجود دارد. سپس ، فرستنده به گيرنده اطلاع می دهد که پیام حاوي گزارشي خاص در قالبی خاص است. اين روش باعث تسهيل ارتباط ماشین هاي مختلف يا اجزاي داخلی مختلف می شود.

هدف اوليه طراحی لایه برنامه کاربردي OSI شامل مجموعه اي از برنامه هاي کاربردي استاندارد شبکه از قبيل پست الکترونيک، انتقال فایل و شبیه سازي ترمینال بود. اما امروزه تبدیل به ظرفی برای گنجاندن تمامی برنامه هاي کاربردي و پروتوكل هايی شده که در لایه هاي زيرين نمي گنجند. از نقطه نظر مدل OSI، تمامی سیستم هاي توزيعي ذاتاً و صرفاً برنامه هاي کاربردي هستند.

آنچه در اين ميان لحاظ نشده، تمایز آشکار بين برنامه هاي کاربردي ، پروتوكلهای کاربردگرا و پروتوكلهای عام منظوره است. بعنوان مثال، پروتکل تبادل فایل اینترنت<sup>۴۹۴</sup> (FTP) (Horowitz and Lunt 1998؛ Postel and Reynolds 1998)، پروتکلي جهت تبادل فایلها بين ماشین مشتری و سرور تعریف می کند. اين پروتکل را نباید با برنامه ftp یکی دانست؛ برنامه مذکور برنامه کاربردي کاربر نهایي جهت انتقال فایل ها است که تصادفاً پروتکل FTP اینترنت را اجرا می کند.

بعنوان نمونه ديگري از انواع پروتکل هاي کاربردگرا می توان به پروتکل انتقال ابرمن<sup>۴۹۵</sup> (HTTP) اشاره کرد (Fielding 1991، همکاران) که جهت مدیریت و هدایت انتقال صفحات وب ازراه دور

<sup>492</sup> session layer

<sup>493</sup> checkpoint

<sup>494</sup> File Transfer Protocol

<sup>495</sup> Hypertext Transfer Protocol

طراحی شده است. این پروتکل بوسیله برنامه های کاربردی از قبیل جستجوگرهاي وب و سرورهاي وب اجرا می شود. با اين وجود، HTTP امروزه در سیستم هایی هم که ذاتاً ارتباطی با وب ندارند، به کار برده می شود. بعنوان مثال، مکانیزم فرآخوانی شیء از راه دور Java از HTTP جهت درخواست فرآخوانی اشیاء از راه دور استفاده می کند که تحت پشتیبانی دیواره آتش قرار دارند (Sun Microsystems, ۲۰۰۴).

پروتکل های عام منظوره متعدد دیگری هم وجود دارند که برای بسیاری از برنامه های کاربردی مناسب هستند، اما نمی توان آنها را در ردیف پروتکل های انتقال قرار داد، بلکه بیشتر بعنوان پروتکل های میان افزار تلقی می شوند. به همین دلیل بحث در مورد آنها را به بعد موکول می کنیم.

### پروتکل های میان افزار

میان افزار، کاربردی است که (در اغلب موارد) منطقاً در لایه برنامه کاربردی قرار می گیرد، اما حاوی تعداد زیادی از پروتکلهای عام منظوره ای است که لایه ای مخصوص به خود و مستقل از کاربردهای مشخص را طلب می کنند. قابل ذکر است که بین پروتکلهای ارتباطات سطح بالا و پروتکل های مربوط به ایجاد سرویسهای مختلف میان افزاری تفاوت وجود دارد.

پروتکل های متعددی جهت پشتیبانی از سرویسهای میان افزاری وجود دارد. بعنوان مثال، همانطور که در فصل<sup>۹</sup> هم خواهیم گفت، روشهای مختلفی برای تصدیق هویت، یعنی اثبات هویت ادعا شده، وجود دارد. پروتکل های تصدیق با هیچیک از برنامه های کاربردی وابستگی خاص نداشته و به همین دلیل می توان آنها را به عنوان یک سرویس عمومی در سیستم میان افزاری تلفیق نمود. به همین ترتیب، آنسته از پروتکل های صدور مجوز که فرآیندها و کاربران تصدیق شده به کمک آنها فقط به منابعی که مجوز آن را دارند دسترسی پیدا می کنند، تمایل دارند که ماهیت عمومی و مستقل از برنامه کاربردی داشته باشند.

بعنوان نمونه دیگر، تعدادی از پروتکل های تعهد توزیعی<sup>۹۶</sup> را در فصل ۸ بررسی خواهیم کرد. پروتکل های تعهد اثبات می کند که در گروهی از فرآیندها، یا تمامی فرآیندها یک عملیات خاص را انجام می دهند یا اینکه اصلاً هیچ عملیاتی انجام نمی شود. این پدیده اتمی بودن<sup>۹۷</sup> هم خوانده می شود و کاربرد گسترده ای در تبادل ها<sup>۹۸</sup> یافته است. همان طور که بعداً خواهیم دید، علاوه بر تبادل ها، برنامه های کاربردی دیگری از قبیل برنامه های تحمل خرابی هم می توانند از پروتکل های تعهد توزیعی بهره مند شوند.

بعنوان آخرین نمونه، پروتکل های قفل توزیعی را در نظر بگیرید که می توان براساس آن و به کمک مجموعه ای از فرآیندها که در ماشین های مختلف توزیع شده اند، از یک منبع در برابر دسترسی های همزمان محافظت نمود. در فصل<sup>۹</sup> تعدادی از این پروتکل ها را بررسی خواهیم کرد. این نیز مثالی از پروتکلی است که از آن می

<sup>496</sup> distributed commit protocols

<sup>497</sup> atomicity

<sup>498</sup> transaction

توان جهت پیاده سازی یک سرویس عمومی میان افزاری استفاده نمود، در عین حال بسیار مستقل از هر برنامه کاربردی خاصی است. پروتکل های ارتباطات میان افزاری از سرویس های ارتباطات سطح بالا پشتیبانی می کنند. بعنوان مثال، در دو بخش بعد راجع به پروتکل هایی بحث خواهیم کرد که باعث می شود تا فرآیند به روشنی بسیار شفاف، رویه مفروضی را فراغوانی یا شیء روی ماشین دوری را فراغوانی کند. به همین ترتیب، سرویس های ارتباطی سطح بالایی هم برای ایجاد و همگام سازی جریانات<sup>499</sup> جهت انتقال داده های زمان دار وجود دارد که بعنوان نمونه در برنامه های کاربردی چندرسانه ای استفاده دارند. مثلاً برخی سیستم های میان افزاری خدمات چندپیخشی قابل اطمینانی را ارائه می دهد که به هزاران گیرنده پراکنده در سرتاسر یک شبکه وسیع توسعه یابد. برخی از پروتکل های ارتباطات میان افزار را می توان به لایه انتقال هم نسبت داد، اما ممکن است دلایل خاصی برای نکه داشتن آنها در سطحی بالاتر وجود داشته باشد. بعنوان مثال، سرویس های چندپیخشی مطمئنی که مقیاس پذیری را تضمین می کند فقط در صورتی قابل پیاده سازی هستند که نیازمندی های برنامه کاربردی را در نظر بگیرند. در نتیجه، یک سیستم میان افزاری ممکن است پروتکل های مختلف (قابل تنظیم) را ارائه دهد که هرچند هریک با استفاده از پروتکلهای انتقالی مختلفی پیاده سازی شده، یک رابط واحد را ارائه می دهد.

**شکل ۴-۳-۱** مدل مرجع تغییریافته برای ارتباطات شبکه ای شده (داخل شکل: برنامه کاربردی، پروتکل انتقال، شبکه، پروتکل فیزیکی، شبکه).

استفاده از این روش در لایه بندي منجر به تولید مدل مرجع نسبتاً سازگاري جهت ارتباطات (مطابق شکل ۴-۳) می شود. در قیاس با مدل OSI، به جای لایه ارائه و جلسه از یک لایه میان افزاری جداگانه استفاده شده که حاوی پروتکل های مستقل از برنامه کاربردی است. این پروتکل ها متعلق به لایه های زیرین پیشتر گفته شده نیست. سرویس های اصلی انتقال ممکن است بدون هیچ اصلاح و تغییری بعنوان یک سرویس میان افزاری هم ارائه شوند. این روش تا حدودی مشابه ارائه UDP در لایه انتقال است. به همین ترتیب، سرویس های ارتباطات میان افزار ممکن است حاوی سرویسهای ردکردن پیام قابل قیاس با سرویس های لایه انتقال باشند.

در ادامه فصل، روی چهار سرویس ارتباطات میان افزار سطح بالا یعنی فراغوانی روال از راه دور(RPC)، سرویس های صف بندي پیام، پشتیبانی از ارتباطات پیوسته رسانه ای از طریق جریانات، و چندپیخشی ها تمرکز خواهیم کرد. اما پیش از آن، معیارهای کلی دیگری برای تمایز ارتباطات (میان افزاری) وجود دارد که ذیلاً در مورد آنها بحث خواهیم کرد.

## ۴-۱-۲- انواع ارتباطات

جهت درک روش های مختلف ارتباطی که میان افزارها برای برنامه های کاربردی ایجاد می کنند، بهتر است مطابق شکل ۴-۴ میان افزار را به عنوان یک سرویس افزوده در محاسبه مشتری- سرور

<sup>499</sup> streams

تصویر نمود. برای مثال، یک سیستم پست الکترونیک را درنظر بگیرید. بطور کلی، هسته سیستم تحویل پست را می‌توان یک سرویس ارتباطی میان افزاری تلقی نمود. هر میزبان با اجرای نماینده کاربر، به او امکان می‌دهد تا پست الکترونیک خود را ساخته، ارسال و دریافت نماید. نماینده کاربر ارسال کننده پست الکترونیک را به سیستم تحویل پست انتقال داده و انتظار دارد که آنهم مقابلاً و در نهایت پیام پستی را به دریافت کننده مورد نظر تحویل دهد. به همین ترتیب، نماینده کاربر طرف دریافت کننده، به سیستم تحویل پیام پستی متصل شده و پیامهای ورودی احتمالی را چک می‌کند. این پیامها در صورت وجود، به نماینده کاربر انتقال داده می‌شوند تا کاربر آنها را مشاهده کرده و بخواند.

سیستم پست الکترونیک نمونه بارزی از ارتباطات پایدار است. در ارتباطات پایدار<sup>۲۰</sup>، پیام تسلیمی جهت ارسال بوسیله میان افزار ارتباطی ذخیره شده و مادامیکه میان افزار اقدام به انتقال آن به گیرنده نکند، در همانجا باقی خواهد ماند. در این حالت، مطابق شکل ۴-۴، پیام توسط میان افزار در یک یا چند وسیله ذخیره سازی، ذخیره می شود. در نتیجه، دیگر لازم نیست تا برنامه کاربردی ارسال کننده پس از تسلیم پیام هم به اجرا ادامه دهد. همچنین لزومی ندارد که پس از تسلیم پیام، برنامه کاربردی دریافت کننده در حال اجرا باشد.

بالعكس، در ارتباطات ناپایدار<sup>۵۰</sup>، پیام فقط تا زمانی توسط سیستم ارتباطی ذخیره می شود که برنامه کاربردی دریافت و ارسال در حال اجرا باشند. به بیان دقیق‌تر، براساس شکل ۴-۴، در صورت وقفه در روند ارسال یا غیرفعال شدن گیرنده، میان افزار دیگر قادر به تحويل پیام نبوده و پیام حذف خواهد شد. غالباً تمامی سرویس‌های ارتباط سطح انتقال فقط ارتباطات ناپایدار را ارائه می دهند. در این حالت، سیستم ارتباطی شامل مسیریاب‌های ذخیره - و - ارسال متداول است. چنانچه یک مسیریاب نتواند پیام را به مسیریاب دیگر یا به میزبان مقصد تحويل دهد، پیام را حذف خواهد کرد.

روش ارتباط علاوه بر دو حالت پایدار و ناپایدار فوق، ممکن است ناهمگام<sup>۰۰</sup> و یا همگام<sup>۰۱</sup> هم باشد. ویژگی بارز ارتباط ناهمگام آن است که فرستنده، آن‌ا پس از تسلیم پیام خود، به کار ادامه می‌دهد. بدین معنا که پیام به محض تسلیم فوراً (وبه طور موقت) در ارتباط همگام<sup>۰۰</sup>، مدامی که میان افزار ذخیره می‌شود.

<sup>500</sup> persistent communication

### 501 transient communication

502 asynchronous

<sup>503</sup> asynchronous  
synchronous

<sup>504</sup> synchronous  
asynchronous communication

<sup>505</sup> asynchronous communication  
synchronous communication

درخواست فرستنده پذیرفته نشود، فرستنده مسدود باقی خواهد ماند. به طور مشخص سه نقطه برای همگام سازی وجود دارد. در حالت اول فرستنده تا زمانی مسدود می‌ماند که میان افزار اطلاع دهد که انتقال این درخواست را انجام می‌دهد. در حالت دوم، فرستنده ممکن است تا زمان تحويل درخواست خود به دریافت کننده مورد نظر، کار همگام سازی را ادامه می‌دهد. در حالت سوم، همگام سازی با انتظار فرستنده برای پردازش کامل درخواست آن یعنی تازمانیکه دریافت کننده پاسخی را بازگرداند، انجام می‌شود.

در عمل، انواع روش‌ها از دیدگاه پایداری و همگامی ممکن است با هم ترکیب شوند که از جمله معروف‌ترین آنها می‌توان به ترکیب حالت پایدار و همگام جهت تسلیم درخواست اشاره کرد. این طرح در بسیاری از سیستم‌های متعارف صفت‌بندی پیام استفاده می‌شود و از این رو در بخش‌های بعد مورد بحث قرار خواهد گرفت. به همین ترتیب، ترکیب ارتباطات ناپایدار و همگام سازی پس از پردازش کامل درخواست بطور گسترده استفاده خواهد شد. این طرح با فرآخوانی روال از راه دور همخوانی دارد که ذیلاً به آن اشاره خواهیم کرد.

علاوه بر پایداری و همگامی، لازم است از تفاوت ارتباطات گستته و جریانی هم صحبت کنیم. تا به اینجا همه مثال‌ها در دسته ارتباطات گستته قرار داشتند. بطوريکه طرفین با پیام با یکدیگر ارتباط برقرار کرده و هر پیام یک واحد کامل اطلاعاتی را در بر داشت. در حالیکه در جریان، پیام‌های متعدد یکی پس از دیگری ارسال شده و بر اساس ترتیب ارسال یا رابطه ای زمانی به یکدیگر مرتبط می‌شوند. در قسمت بعد مفصلًاً راجع به این موضوع بحث خواهیم کرد.

#### ۴-۲- فرآخوانی روال از راه دور (RPC)

بسیاری از سیستم‌های توزیعی بر اساس تبادل آشکار پیام بین فرآیندها استوار هستند. با این وجود، رویه‌های *receive* و *send* نمی‌توانند ارتباط را به هیچ وجهی پنهان سازند و این ویژگی تأثیر مهمی در به دست آوردن شفافیت دسترسی در سیستم‌های توزیعی دارد. این اشکال مدت‌هاست که شناسایی شده، اما تلاش چندانی برای رفع آن صورت نگرفته بود تا اینکه Birrell and Nelson با ارائه مقاله‌ای در سال ۱۹۸۴ روش کاملاً متفاوتی را برای مسئله ارتباط ارائه دادند. هر چند ایده آنها (در ظاهر) بسیار ساده می‌نماید، اثر جانبی ظرفی دارد. در این بخش، مفهوم، اجرا، مزایا و معایب آنرا بررسی خواهیم کرد.

به طور خلاصه، Birrell and Nelson پیشنهاد کردند که برنامه‌ها بتوانند رویه‌های ماشین‌های دیگر را فرآخوانی نمایند. هنگامی که فرآیند ماشین *A* اقدام به فرآخوانی روال ماشین *B* می‌کند، فرآیند فرآخواننده در *A* معلق شده و اجرای رویه فرآخوانی شده روی ماشین *B* انجام می‌شود. اطلاعات را می‌توان به صورت پارامتری از فرآخواننده به فرآخوان شده انتقال داد و به شکل نتیجه رویه بازگرداند. بعلاوه، برنامه نویس به هیچ وجه قادر به

مشاهده ردکردن پیام ها نخواهد بود. این روش به نام فراخوانی روال از راه دور یا **RPC** خوانده می شود.

هرچند ایده اساسی بسیار ساده و بی نقص به نظر می رسد، مشکلات ظریفی در این میان وجود دارد. به عنوان پیش زمینه به خاطر داشته باشد از آنجایی که روال های فراخوانی و فراخوان شده روی ماشین های مختلف اجرا می شوند، فضاهای آدرسی مختلفی دارند که باعث پیچیدگی می شود. پaramترها و نتایج نیز باستی ردوبدل شوند که اینهم، درصورت تفاوت ماشین ها ، باعث افزایش پیچیدگی خواهد شد . نهایتاً هر یک از دو ماشین می توانند چار شکست شوند و حالت های مختلف شکست مسائل مختلفی را ایجاد می کنند. اما غالب این مسائل قابل حل بوده و **RPC** شیوه پرکاربردی است که شالوده بسیاری از سیستمهای توزیعی را تشکیل می دهد.

#### ٤-٢-١ - عملیات اصلی **RPC**

بحث خود را با فراخوانی روال مرسوم آغاز کرده و سپس توضیح می دهیم که چگونه می توان خود فراخوانی را به صورت اجزاء مشتری و سرور تجزیه نمود که هر یک روی ماشین های مختلف قابل اجراست.

#### فراخوانی روال مرسوم

برای درک نحوه عملکرد **RPC** ، درک کامل عملکرد فراخوانی روال مرسوم (یعنی فقط یک ماشین) اهمیت خاصی دارد. یک فراخوانی در C را به این صورت فرض کنید:

```
count =read(fd , buf, nbytes )
```

که در آن *fd* عدد صحیح بیانگر فایل، *buf* رشته اي از کاراکترهایی است که داده ها درآن خوانده می شود و *nbytes* عدد صحیح دیگری است که بیانگر تعداد بایتهاي خواندنی است. چنانچه از برنامه اصلی فراخوانی شود، پشته<sup>٥٠٦</sup> پیش از فراخوانی به صورت شکل (الف) ٤-٥ خواهد بود. بر اساس شکل (ب) ٤-٥ ، فراخواننده جهت فراخوانی، پaramترها را بر روی پشته به صورت آخرین ورود / اولین خروج قرار خواهد داد. (دلیل آنکه کمپایلرهای C پaramترها را به ترتیب عکس وارد می کنند مربوط به *printf* است. چون به این ترتیب، *printf* قادر خواهد بود تا اولین پaramتر خود یعنی زنجیره  *قالب*<sup>٥٠٧</sup> را مکان یابی نماید). پس از اتمام اجرای روال *read*، مقدار بازگشت در یک ثبات<sup>٥٠٨</sup> قرار داده شده ، آدرس بازگشت حذف می شود و کنترل را به فراخواننده برミ گرداند. سپس ، فراخواننده پaramترها را از پشته جدا کرده و پشته را به وضعیت اصلی پیش از فراخوانی بازمی گرداند.

شکل ٤-٥- الف) رد کردن پaramتر فراخوانی روال محلی: پشته پیش از فراخوانی **read**. ب) پشته در حین فعل بودن روال فراخوان شده (داخل شکل: الف) متغیرهای محلی برنامه اصلی، نشانگر پشته<sup>٥٠٩</sup>، ب) متغیرهای محلی برنامه اصلی، آدرس بازگشت، متغیرهای محلی **(read**

<sup>506</sup> stack

<sup>507</sup> Last in / First out (LIFO)

<sup>508</sup> format string

<sup>509</sup> registerer

<sup>510</sup> stack pointer

چند نکته را باید خاطرنشان نمود. در درجه اول، در C پارامترها ممکن است **فراخوانی توسط مقدار<sup>۱۱۰</sup>** یا **فراخوانی توسط مرجع<sup>۱۱۱</sup>** باشند. همانطور که در شکل (ب) ۴-۵ مشاهده می کنید، پارامتر مقداری از قبیل *nbytes* یا *fd* به پشته کپی می شوند. از نظر روال فراخوانی شده، پارامتر مقدار صرفاً یک متغیر محلی مقدارداده شده است. روال فراخوانی شده می تواند آنرا تغییردهد، اما چنین تغییراتی هیچ تأثیری بر مقدار اولیه در طرف فراخواننده نمی گذارد.

هر پارامتر مرجع در C، نشانگری به یک متغیر (یعنی آدرس متغیر) است، نه به مقدار آن. در حالت فراخوانی *read*، از آنجاییکه آرایه ها<sup>۱۱۲</sup> همواره بوسیله مرجع در C رد کرده می شوند، پارامتر دوم یک پارامتر مرجع خواهد بود. آنچه واقعاً به پشته وارد می شود، آدرس آرایه کاراکتری است. چنانچه روال فراخوانده شده از این پارامتر جهت ذخیره سازی در آرایه کاراکتری استفاده کند، قطعاً آرایه را در روال فراخوانی شده اصلاح خواهد کرد. همچنانکه بعداً خواهیم دید، تفاوت بین فراخوانی توسط مقدار و فراخوانی توسط مرجع اهمیت خاصی در RPC دارد.

مکانیزم رდکردن پارامتر دیگری هم به نام **فراخوانی توسط کپی/بازیابی<sup>۱۱۳</sup>** وجود دارد که در C قادر کاربرد است. این مکانیزم شامل کپی کردن متغیر به پشته توسط فراخواننده است (که از این نظر مشابه فراخوانی توسط مقدار می باشد). متغیر پس از خاتمه فراخوانی کپی و بازگردانده می شود و روی مقدار اولیه فراخواننده رونویسی می شود. در اغلب موارد، تأثیر ایجاد شده دقیقاً مشابه تأثیر فراخوانی توسط مرجع است. در برخی موارد، از جمله زمانیکه پارامتر واحد چندین مرتبه در لیست پارامترها قرار داده می شود، مفهوم دیگری برای خواندن آن مشاهده می شود. اما در بسیاری از زبانها از مکانیزم فراخوانی توسط کپی/بازیابی استفاده نمی شود.

تصمیم گیری راجع به روش مناسب رდکردن پارامتر غالباً بر عهده طراحان زبان بوده و ویژگی ثابت زبان محسوب می شود. این امر گاهی اوقات وابسته به نوع داده رد کرده می باشد. بعنوان مثال، همچنانکه بعداً خواهیم دید، در C اعداد صحیح و دیگر انواع غیربرداری<sup>۱۱۴</sup> همواره توسط مقدار رد می شوند، در حالیکه آرایه ها، توسط مرجع. برخی از کمپایلرهای Ada برای پارامترهای **in out**، از کپی/بازیابی استفاده می کنند، اما در برخی دیگر از فراخوانی توسط مرجع استفاده می شود. نحوه تعریف زبان امکان انتخاب یکی از این دو حالت را فراهم می کند که البته معنی را هم تا حدودی پیچیده و مبهم می کند.

## مشتری مجازی<sup>۱۱۵</sup> و سرور مجازی<sup>۱۱۶</sup>

<sup>۱۱۱</sup> call by value

<sup>۱۱۲</sup> call by reference

<sup>۱۱۳</sup> array

<sup>۱۱۴</sup> copy/restore

<sup>۱۱۵</sup> scalar

<sup>۱۱۶</sup> client stub

ایده زیربنایی **RPC** این است که فرآخوانی روال از راه دور حتی الامکان به صورت فرآخوانی روال محلی به نظر برسد. به بیان دیگر، **RPC** باید شفاف باشد و روال فرآخوانی نباید از اجرای روال فرآخواننده شده روی ماشین دیگر اطلاع داشته باشد و بر عکس. فرض کنید که بخواهیم برنامه ای بعضی از داده ها از یک فایل را بخواند. برنامه نویس برای کسب داده ، فرآخوان *read* را در کد خود وارد می کند. در سیستم های متعارف (تک پردازشگری) ، روال *read* توسط پیوند دهنده<sup>۵۱۸</sup> از کتابخانه استخراج و داخل برنامه مورد نظر درج می شود. این روال کوتاه غالباً از طریق فرآخوانی یک فرآخوان سیستمی معادل برای *read* پیاده سازی می شود. به بیان دیگر، روال *read* نوعی رابط بین کد کاربر و سیستم عامل محلی است.

مطابق شکل (ج) ۴-۵، اگرچه *read* اقدام به فرآخوان سیستمی می نماید، ولی به همان صورت عادی یعنی با قراردادن پارامترها روی پشته فرآخواننده خواهد شد. به این ترتیب، برنامه نویس کوچکترین اطلاعی از این عمل زیرکانه *read* پیدا نخواهد کرد.

**RPC** شفافیت خود را به روش مشابه کسب می کند. در صورتیکه *read* واقعاً روال از راه دور باشد، (مثلًاً روالی که روی ماشین سرور اجرا خواهد شد)، نسخه متفاوتی از *read* به نام **مشتری مجازی** در کتابخانه قرار داده خواهد شد. این نسخه هم مشابه *read* اصلی با استفاده از زنجیره فرآخوانی شکل (ج) ۴-۵ فرآخوانده می شود . همچنین مشابه *read* اصلی سیستم عامل محلی را فرآخوانی می کند. تنها تفاوت این است که برخلاف اصل خود از سیستم عامل تقاضای گرفتن داده را نخواهد کرد، بلکه در عوض پارامترها را به صورت پیام بسته بندی کرده و درخواست می کند که پیام مطابق شکل ۴-۶ به سرور ارسال شود. پس از فرآخوانی *send*، مشتری مجازی را فرآخوانی کرده و تا زمان رسیدن پاسخ، خود را بلوکه می کند. شکل ۴-۶ - اصل کلی **RPC** بین برنامه مشتری و سرور (داخل شکل: مشتری، انتظار دریافت نتیجه، فرآخوانی روال از راه دور، بازگشت از فرآخوانی، درخواست، پاسخ، سرور، فرآخوانی روال محلی و بازگشت نتایج، زمان).

با رسیدن پیام به سرور، سیستم عامل سرور آنرا به سرور مجازی انتقال می دهد. سرور مجازی در طرف سرور معادل یک **مشتری مجازی** است. به بیان دیگر ، قطعه ای از یک کد است که درخواستهای ورودی از سرتاسر شبکه را به فرآخوان های روال محلی تبدیل می کند. معمولاً سرور مجازی *receive* را فرآخوانی نموده و تا رسیدن پیامهای ورودی بلوکه باقی می ماند. سرور مجازی پارامترهای پیام را باز کرده ، سپس مطابق معمول (شکل ۴-۵) روال سرور را فرآخوانی می نماید. از نقطه نظر سرور، این وضعیت مشابه فرآخوانی مستقیم سرور بوسیله مشتری است - پارامترها و آدرس بازگشت همگی روی پشته ای قرار دارند که به آن متعلق بوده و همه چیز حالت عادی خواهد داشت. سرور کار خود را انجام داده و سپس نتیجه را به صورت معمول به فرآخوان بازمیگرداند. بعنوان

<sup>۵۱۷</sup> server stub

<sup>۵۱۸</sup> linker

مثال، در مورد *read* ، سرور با فری<sup>۵۱۹</sup> را که به توسط پارامتر دوم به آن اشاره شده است را پر از داده می نماید، این با فرداخلي سرور مجازي خواهد بود.

پس از تكميل فرداخلي، هنگام يك سرور مجازي كنترل را مجدداً در دست مي گيرد، نتيجه (با فر) را در قالب يك پيام بسته بندی كرده و *send* را فرداخلي می کند تا آنرا به مشتري بازگرداند. عموماً پس از آن ، سرور مجازي مجدداً *receive* را فرداخلي می کند تا منتظر درخواست ورودي بعد بماند.

پس از آنکه پيام به ماشين مشتري بازگرداند می شود، سистем عامل مشتري مشاهده می کند که به فرآيند مشتري (يا مشتري مجازي) آدرس دهي شده است (توجه داشته باشيد که از نظر سистем عامل ايندو تفاوتی با هم ندارند). پيام به با فر درحال انتظار کپي شده و فرآيند مشتري آزاد می شود. مشتري مجازي پيام را بازبیني كرده، نتيجه را باز كرده، آنرا به فرداخلي اش کپي و به همان صورت معمول بازمي گرداند . پس از فرداخلي *read* ، هنگامي که فرداخلي اند كنترل را مجدداً در اختيار مي گيرد ، فقط از وجود داده ها اطلاع دارد و كوچکترین اطلاعی ندارد که در عوض سистем عامل محلی، اينكار از راه دور انجام شده است.  
اما اين ناآگاهي مشتري، در واقع راز ظرافت و جذابيت کلي طرح است . تا به اينجا ، سرويس هاي از راه دور از طريق ايجاد فرداخلي هاي روالي عادي (يعني محلی) قابل دسترس بودند، نه از طريق فرداخلي *send* و *receive*.. همان گونه که جزئيات انجام فرداخان سистемي در كتابخانه هاي متعارف مخفی هستند، تمامي جزئيات مربوط به رديكندن پيام در دو روالي كتابخانه قرار داده می شوند .  
بطور خلاصه، فرداخلي روالي از راه دور طي مراحل زير انجام می شود :

۱. روالي مشتري، مشتري مجازي را به طور معمول فرا مي خواند.
۲. مشتري مجازي پس از ساخت پيام، سистем عامل محلی را فرامي خواند.
۳. OS (سيستم عامل) مشتري پيام را به OS ماشين دور ارسال مي کند.
۴. OS ماشين دور پيام را به سرور مجازي مي دهد.
۵. سرور مجازي پارامتر ها را باز كرده و سرور را فرداخلي مي کند.
۶. سرور کار خود را انجام داده نتيجه را به سرور مجازي بازمي گرداند.
۷. سرور مجازي نتيجه را به شكل پيام بسته بندی نموده و OS محلی خود را فرا مي خواند.
۸. OS سرور پيام را به OS مشتري ارسال مي کند.
۹. OS مشتري پيام را به مشتري مجازي مي دهد.
۱۰. مشتري مجازي نتيجه را باز و آنرا به مشتري بازمي گرداند.

<sup>519</sup> buffer

نتیجه خالص مراحل فوق، تبدیل فرآخوانی محلی توسط روال مشتری به مشتری مجازی، به فرآخوانی محلی به روال سرور است، بدون آنکه مشتری یا سرور حتی از مراحل میانی یا وجود شبکه کوچکترین اطلاعی داشته باشد.

#### ۴-۲-۲ - رد کردن پارامتر<sup>۵۲۰</sup>

وظیفه مشتری مجازی، گرفتن پارامترهای مربوط به خود، بسته بندی آنها به شکل پیام و ارسال به سرور مجازی است. اینکار برخلاف ظاهر سهل و ساده خود دارای پیچیدگی های خاصی است. در این بخش، نگاهی به برخی از مسائل مرتبط با ردکردن پارامتر در سیستم های RPC خواهیم داشت.

#### رد کردن پارامترهای مقدار

بسته بندی پارامترها به شکل پیام، مرتب سازی پارامتر<sup>۵۲۱</sup> نامیده می شود. برای نمونه یک روال از راه دور ساده مثل اضافه کردن (ز و ن) را در نظر بگیرید. در این حالت، دو پارامتر عدد صحیح ز و ن گرفته شده و حاصل جمع ریاضی آنها بصورت نتیجه بازگردانده می شود (البته در عمل هیچکس چنین روال ساده از راه دوری را انجام نمی دهد، ولی برای مثال مناسب است). فرآخوانی اضافه کردن در سمت چپ شکل ۴-۷ (در بخش فرآیند مشتری) نمایش داده شده است. مشتری مجازی ایندو پارامتر را گرفته و آنها را به ترتیب گفته شده به پیام تبدیل می کند. سپس، نام یا شماره روالی را که قرار است فرآخوانی شود، در پیام قرار می دهد. از آنجاییکه امکان دارد سرور چندین فرآخوان را پشتیبانی نماید، فرآخوان مورد نظر باید مشخص شود.

شکل ۴-۷ - مراحل مربوط به انجام محاسبه از راه دور در RPC (داخل شکل: ماشین مشتری، فرآیند مشتری، OS مشتری، فرآخوانی روال توسط مشتری، مشتری مجازی، سرور مجازی، مشتری مجازی پیام را ایجاد می کند، ماشین سرور، فرآیند سرور، سرور مجازی فرآخوانی محلی add را انجام می دهد، سرور مجازی پیام را باز می کند، OS سرور پیام را به سرور مجازی تحويل می دهد).

وقتی پیام به سرور می رسد، سرور مجازی پس از بررسی، روال لازم را شناسایی و فرآخوانی مناسب را انجام می دهد . چنانچه سرور روال های از راه دور دیگری را هم پشتیبانی کند، ممکن است سرور مجازی - برحسب فیلد اول پیام - از جمله سوئیچ برای انتخاب روال فرآخوانده شده استفاده کند. فرآخوانی واقعی از سرور مجازی به سرور مشابه فرآخوانی مشتری اصلی است، تنها تفاوت در اینجاست که پارامترها، متغیرهایی هستند که از پیام وارد آغاز می شوند. پس از اتمام کار سرور، کنترل مجدداً در اختیار سرور مجازی قرار می گیرد. همچنین نتیجه عودت داده شده توسط سرور را اخذ و آنرا به شکل پیام دسته بندی می کند. این پیام مجدداً به ماشین مشتری ارسال و توسط آن باز می شود. مشتری مجازی نتیجه را استخراج و مقدار را به روال مشتری در حال انتظار باز می گرداند.

مدادامی که ماشین های مشتری و سرور یکسان بوده و تمامی پارامترها و نتایج از انواع اسکالر از قبیل اعداد صحیح،

<sup>520</sup> parameter passing

<sup>521</sup> parameter marshaling

کاراکترها و منطقی باشند، این مدل به خوبی عمل خواهد کرد. اما در یک سیستم بزرگ توزیعی، معمولاً چندین نوع ماشین وجود دارد و هر ماشین غالباً از روش خاصی برای نمایش اعداد، کاراکترها و دیگر اقلام داده ای استفاده می کند. بعنوان مثال، در کامپیوترهای بزرگ IBM از کد EBCDIC برای حروف و در کامپیوترهای شخصی IBM از کد ASCII استفاده می شود. در نتیجه، مطابق طرح ساده شده شکل ۴-۷ امکان انتقال یک پaramتر کاراکتری از مشتری کامپیوتر شخصی IBM به سرور کامپیوتر بزرگ IBM وجود ندارد. حتی در صورت انتقال، کاراکتر توسط سرور به صورت نادرستی ترجمه خواهد شد.

نمایش اعداد صحیح ( متمم عدد یک در مقایسه با متمم عدد دو) و اعداد ممیز شناور هم ممکن است با مشکلاتی مشابه همین انجام شود. مشکل بزرگتر این است که برخی ماشینها از جمله Intel Pentium، بایتها را از راست به چپ شماره گذاری می کنند ، در حالیکه برخی دیگر از قبیل Sun SPARK از جهت دیگر برای شماره گذاری استفاده می کنند. با الهام از داستان سفرهای گالیور و درگرفتن جنگ بین سیاستمداران بر سر شکستن تخم مرغ در این کتاب، فرمت Intel سرکوچک<sup>۵۲۲</sup> و فرمت SPARK سربزرگ<sup>۵۲۳</sup> نامیده می شود (Cohen, ۱۹۸۱). بعنوان مثال، یک روال را با دو پaramتر یکی به صورت عدد صحیح و دیگری یک زنجیره چهار حرفی در نظر بگیرید. هر پaramتر نیازمند یک کلمه ۳۲ بیتی است. شکل (الف) ۴-۸ بخش پaramتری پیام ساخته شده توسط مشتری مجازی در Intel Pentium را نمایش می دهد. کلمه اول حاوی پaramتر عدد صحیح ، یعنی ۵، و کلمه دوم حاوی رشته "JILL" است.

**شكل ۴-۸ - الف) پیام اصلی در Pentium . ب) پیام پس از دریافت در SPARK . ج) پیام پس از تبدیل. اعداد ریز داخل مربعها بیانگر آدرس هر بایت است.**

از آنجاییکه پیام ها بایت به بایت ( در عمل، بیت به بیت) در سرتاسر شبکه انتقال داده می شود، اولین بایت ارسالی اولین بایت رسیده خواهد بود. در شکل (ب) ۴-۸ نمایش داده ایم که پیام شکل (الف) ۴-۸ در صورت دریافت بواسیله SPARK چگونه بنظر خواهد رسید. برخلاف تمامی تراشه های Intel که بایت از راست شماره گذاری می شود ( بایت مرتبه پایین)، SPARK اینکار را از بایت صفر در سمت چپ( بایت مرتبه بالا) شروع می کند. اگر سرور مجازی پaramترها را به ترتیب در آدرس های ۰۴ و ۰۵ بخواند، عدد صحیحی برابر با  $(5 \times 2^{24}) + 83886080$  و زنجیره "JILL" بدست خواهد آمد. یک روش آشکار اما متأسفانه نادرست، معکوس کردن ساده ترتیب بایت های هر کلمه پس از دریافت است که منجر به تولید شکل (ج) ۴-۸ خواهد شد. برای این حالت، عدد صحیح ۵ و رشته "JILL" را در نظر بگیرید . مشکل این است که اعداد صحیح از طریق مرتب سازی متفاوتی نسبت به رشته ها معکوس شده اند، اما رشته ها بدون تغییر باقی می مانند. بدون در اختیار داشتن اطلاعات کافی

<sup>522</sup> little endian

<sup>523</sup> big endian

در مورد ماهیت رشته و عدد صحیح، جبران اشتباهات غیرممکن خواهد بود.

### ردکردن پارامترهای مرجع

سؤال این است که نشانگرها<sup>۵۲۴</sup>، یا به طور کلی ، مرجع ها<sup>۵۲۵</sup> چگونه رد می شوند؟ در پاسخ باید گفت که اینکار بسیار مشکل یا در بعضی موارد بطور کلی غیرممکن است. به خاطر داشته باشد که یک نشانگر فقط در داخل فضای آدرس فرآیندی معنا دارد که در آن استفاده شده است. مجدداً مثال خواندن را بیاد بیاورید که در آن چنانچه پارامتر دوم (آدرس بافر) در مشتری ۱۰۰۰ باشد، نمی توان فقط عدد ۱۰۰۰ را به سرور انتقال داد و انتظار نتیجه خوبی را هم داشت. ممکن است آدرس ۱۰۰۰ در سرور در وسط متن برنامه قرار داشته باشد. یک راه حل ساده آن است که تمام نشانگرها و پارامترهای مرجع را ممنوع کرد. اما اهمیت ایندو المان به حدی است که ممنوع سازی آنها هم واقعاً بسیار غیرمنطقی و هم غیرلازم است. در مثال خواندن (read)، مشتری مجازی می داند که پارامتر دوم اشاره به رشته ای از کاراکترها دارد. تصور کنید که بزرگی و ابعاد رشته هم مشخص باشد. بنابراین راهکار احتمالی، کپی کردن رشته در پیام و ارسال آن به سرور است. در اینصورت سرور مجازی می تواند با یک نشانگر به رشته ، سرور را فراخوانی نماید. این امر حتی در صورتی هم که نشانگر دارای مقدار عدی متفاوت از پارامتر دوم read باشد، امکان پذیر است. تغییراتی که سرور با استفاده از نشانگر ایجاد می کند (مانند ذخیره داده ها در داخل آن) تأثیر مستقیمی بر بافر پیام داخل سرور مجازی می گذارد. وقتی کار سرور به پایان رسید، می توان پیام اصلی را به مشتری مجازی بازگردانده و سپس آنرا مجدداً به مشتری کپی نمود. در نتیجه ، روش کپی/ بازیابی جایگزین فراخوانی توسط مرجع می شود. هرچند کاملاً متناظر نیست، ولی در بسیاری از موارد عملکرد قابل قبولی دارد.

اثربخشی این مکانیزم را می توان از طریق بهینه سازی می توان دوچندان کرد. چنانچه مجازی ها بدانند که بافر پارامتر ورودی به سرور است یا خروجی از آن، می توان یکی از کپی ها را حذف نمود. چنانچه آرایه ورودی به سرور باشد (بعنوان مثال در فراخوانی نوشتن) دیگر نیازی به کپی برای برگشت آن نیست. اما چنانچه خروجی باشد، لازم نیست که در مرحله اول فرستاده شود. در پایان لازم به ذکر است که هرچند اکنون قادر به اداره اشاره گرها به آرایه ها و ساختارهای ساده هستیم، هنوز راه حلی برای حالت های عام یعنی اداره اشاره گر به ساختارهای داده ای دلخواهی از قبیل یک گراف پیچیده ارائه نشده است. برخی از سیستم ها تلاش می کنند تا با ردکردن واقعی اشاره گر به سرور مجازی و تولید کد ویژه در روای سرور برای استفاده از نشانگر بر این مشکل فائق آیند. بعنوان مثال، می توان برای تأمین داده ارجاع داده شده ، درخواست را به مشتری بازگرداند.

### توصیف پارامتر و تولید مجازی ها

<sup>524</sup> pointer

<sup>525</sup> reference

از آنچه تاکنون توضیح داده شد به راحتی می‌توان دریافت که مخفی سازی فراخوانی روای را از راه دور مستلزم توافق فراخواننده و فراخوان شده در مورد قالب پیام تبادلی است. بعلاوه بایستی در مواردی از قبیل ردکردن ساختارهای پیچیده داده ای از مراحل یکسانی تبعیت کنند. به بیان دیگر، طرفین **RPC** بایستی از یک پروتکل واحد تبعیت کرده و درغیراینصورت **RPC** عملکرد صحیح نخواهد داشت.

برای نمونه، روای شکل (الف) ۴-۹ را در نظر بگیرید. این روای دارای سه پارامتر، یک کاراکتر، یک عدد صحیح ممیز شناور و آرایه ای متشکل از پنج عدد صحیح است. با این فرض که یک کلمه از چهار بایت تشکیل می‌شود، در پروتکل **RPC** تصریح شده که کاراکتر را در اولین بایت سمت راست کلمه (سه بایت بعدی را خالی بگذارد)، عدد ممیز شناور را به صورت یک کلمه کامل و آرایه را بعنوان گروهی از کلمات با طول برابر با رشته انتقال دهیم و پیش از اینها هم کلمه ای به عنوان بیانگر طول ارسال شود (شکل (ب) ۴-۹). براساس این قوانین، مشتری مجازی **foobar** می‌داند که باید از قالب شکل (ب) ۴-۹ استفاده کرد و سرور مجازی می‌داند که پیامهای ورودی از **foobar** بایستی در قالب شکل (ب) ۴-۹ صدق کند.

#### شکل (الف) ۴-۹ یک روای ب) پیام متناظر روای (داخل شکل: متغیرهای محلی (**foobar**)

هرچند تعریف فرمت پیام یکی از ابعاد لازم پروتکل **RPC** است، اما کافی نمی‌باشد. بعلاوه لازم است که مشتری و سرور در مورد ارائه ساختارهای ساده داده ای از قبیل اعداد صحیح، کاراکترها، اعداد جبر بولی و غیره هم توافق داشته باشند. بعنوان مثال، ممکن است در پروتکل تصریح شود که اعداد صحیح در متمم دو، کاراکترها در استاندارد یونی کد ۱۶ بیتی و اعداد ممیز شناور در فرمت #۷۵۴ IEEE نمایش داده شده و همه چیز در سرکوچک ذخیره شود. با داشتن این اطلاعات اضافی می‌توان پیامها را به وضوح تفسیر نمود.

پس از آنکه قوانین کدگذاری تا بیت آخر مشخص شدند، فقط لازم است که فراخواننده و فراخوان شده در مورد تبادل واقعی پیامها به توافق برسند. بعنوان مثال، ممکن است توافق شود که از سرویس انتقال اتصال گرایی مثل **TCP/IP** استفاده شود. روش دیگر، استفاده از یک سرویس دیتاگرام نامطمئن است. سپس به مشتری و سرور اجازه داد تا بتوان بعنوان بخشی از پروتکل **RPC**، یک طرح کنترل خطاب اجرا کنند. در عمل از راهکارهای متفاوتی استفاده می‌شود.

پس از تعریف کامل و دقیق پروتکل **RPC**، بایستی مشتری و سرور اجرا شود. خوشبختانه مجازی های یک پروتکل واحد با روای های مختلف، غالباً فقط از نظر رابط با برنامه های کاربردی تفاوت دارند. یک رابط از مجموعه ای از روای ها تشکیل شده که بوسیله مشتری فراخوانی شده و توسط سرور اجرا می‌شود. رابط معمولاً به همان زبان برنامه نویسی قابل دسترسی است که سرور و مشتری به آن نوشته می‌شوند (هرچند لزومی به انجام آن نیست). به بیان ساده

تر، رابط‌ها غالباً بوسیله زبان تعریف رابط<sup>۵۲۶</sup> (IDL) تعریف می‌شوند. سپس، این رابط به همراه رابط‌های مناسب زمان کمپایل یا زمان اجرا، به صورت مشتری مجازی و سرور مجازی کمپایل می‌شوند. در عمل اثبات شده که استفاده از زبان تعریف رابط به نحو چشمگیری باعث تسهیل برنامه کاربردی مشتری - سرور بر اساس RPC ها می‌شود. از آنجایی که ساخت کامل مشتری و سرور مجازی امری ساده است، تمامی سیستم‌های میان افزاری براساس RPC جهت حمایت از توسعه برنامه‌های کاربردی، یک IDL ارائه می‌دهند. همچنانکه در بخش‌های بعدی همین فصل مشاهده خواهیم کرد، در برخی از موارد استفاده از IDL الزامی است.

### ٤-٢-٣ RPC ناهمگام

مشابه فرآخوانی‌های روال معمول، هنگامی که مشتری روال دوری را فرامی‌خواند، تا زمان بازگشت پاسخ مسدود باقی خواهد ماند. این رفتار انعطاف ناپذیر درخواست/پاسخ هنگامی غیر ضروري می‌نماید که هیچ نتیجه ای بازگشت داده نشده و فقط منجر به مسدود کردن مشتری شود؛ در حالیکه می‌توانست پیشروی کرده و پس از درخواست فرآخوانی روال از راه دور کار مفیدی انجام دهد. از نمونه مواردی که غالباً نیازی به انتظار برای دریافت پاسخ نیست می‌توان به انتقال پول از یک حساب به حساب دیگر، افزودن لیست به پایگاه داده ای، راه اندازی خدمات از راه دور، پردازش دسته و غیره اشاره کرد.

سیستم‌های RPC جهت پشتیبانی از چنین مواردی امکاناتی به نام RPC های ناهمگام<sup>۵۲۷</sup> ارائه می‌دهند و بدین ترتیب، پس از صدور درخواست RPC، مشتری فوراً فعال می‌شود. در RPC های ناهمگام، سرور به محفوظ دریافت درخواست RPC پاسخی را برای مشتری ارسال کرده و سپس روال درخواست شده را فرا می‌خواند. این پاسخ برای مشتری به عنوان تصدیقی است که سرور قصد پردازش RPC را دارد. مشتری به محفوظ دریافت تصدیق سرور، مسدود شدن را پایان داده و شروع به کار خواهد کرد. شکل (ب) ٤-١٠ نحوه تقابل مشتری- سرور را در RPC های ناهمگام نمایش می‌دهد. برای سهولت مقایسه، در شکل (الف) ٤-١٠ رفتار عادی درخواست/پاسخ نمایش داده شده است.

RPC های ناهمگام علاوه بر این درحالی می‌توانند مفید باشند که پاسخ بازگردانده خواهد شد، اما مشتری آمادگی برای انتظار کشیدن و غیرفعال ماندن در این فاصله زمانی ندارد. بعنوان مثال، مشتری ممکن است بخواهد آدرس‌های شبکه ای مجموعه ای از میزبان‌هایی را که انتظار می‌رود در آینده تماس بگیرند، پیش‌واکشی کند. منطق حکم می‌کند که در چنین مواردی ارتباط بین مشتری و سرور از طریق دو RPC ناهمگام مطابق شکل ٤-١١ سازماندهی شود. ابتدا مشتری سرور را فرآخوانی می‌کند تا لیستی از اسامی میزبانان قابل جستجو را ارائه دهد. پس از تصدیق دریافت لیست توسط سرور، مشتری به کار خود ادامه می‌دهد. فرآخوانی دوم توسط سروری انجام می‌شود که مشتری را برای تحويل آدرس‌های یافت شده

<sup>526</sup> Interface Definition Language

<sup>527</sup> asynchronous RPCs

فراخوانی می کند. ترکیب دو RPC ناهمگام در بعضی موارد به نام **RPC همگام معوق**<sup>۵۲۸</sup> هم نامیده می شود.

لازم به ذکر است که در برخی از انواع RPC های ناهمگام ، مشتری فوراً پس از ارسال درخواست به سرور، همچنان به اجرا ادامه می دهد. به بیان دیگر، مشتری منتظر تصدیق پذیرش درخواست سرور باقی نمی ماند. ما چنین RPC های یکطرفه<sup>۵۲۹</sup> می نامیم. مشکل این روش آن است که در صورتی که قابلیت اطمینان تضمین نشود، مشتری بطور قطع نمی داند که آیا درخواست او پردازش خواهد شد یا نه. در فصل ۸ راجع به این مسائل بحث خواهیم کرد. به همین ترتیب، در RPC های همگام معوق، مشتری می تواند به جای این که به سرور اجازه دهد نتیجه را بعداً به صورت فراخوان به مشتری ارسال کند، در مورد آماده بودن نتیجه از سرور پرس و جو کند.

شكل ۴-۱۰ - الف) تقابل بین مشتری و سرور در RPC متداول.ب) تقابل با استفاده از RPC ناهمگام (داخل شکل: انتظار برای نتیجه، مشتری، روال فراخوانی از راه دور، بازگشت از فراخوانی، درخواست، پاسخ، سرور، زمان، روال فراخوانی محلی و بازگشت نتایج، انتظار برای پذیرش، مشتری، روال فراخوانی از راه دور، بازگشت از فراخوانی، درخواست، پذیرش درخواست، سرور، زمان، روال فراخوانی از راه دور).

شكل ۴-۱۱ - تبادل مشتری و سرور از طریق دو RPC ناهمگام (داخل شکل: انتظار برای نتیجه، مشتری، مشتری، روال فراخوانی از راه دور، بازگشت از فراخوانی، درخواست، پذیرش درخواست، سرور، روال فراخوانی از راه دور، وقفه مشتری، بازگشت نتایج، اعلان وصول، زمان، فراخوانی مشتری بوسیله RPC یکطرفه).

## ۲-۴-۲ - مثال: DCE PRC

از فراخوانی های از راه دور بطور گستردگی بعنوان مبنای سیستم های توزیعی و میان افزاری استفاده شده است. در این بخش، نگاه دقیقتری به یکی از سیستم های ویژه RPC خواهیم داشت. این سیستم که **محیط محاسبه توزیعی (DCE)**<sup>۵۳۰</sup> نامیده میشود، توسط OSF<sup>۵۳۱</sup> ابداع و ارائه شد، اما امروزه بیشتر از آن بعنوان گروه باز<sup>۵۳۲</sup> یاد می شود. آوازه DCE RPC به اندازه برخی دیگر از سیستم های RPC از جمله Sun RPC نیست. با این وجود، DCE PRC می تواند نماینده دیگر سیستم های PRC باشد و در سیستم مبنای مایکروسافت از ویژگی های آن جهت محاسبه توزیعی، DCOM ، استفاده شده است (Eddon and Eddon ۱۹۹۸). بحث خود را با تعریف کوتاهی از DCE شروع کرده و سپس وظایف اصلی DCE RPC را شرح می دهیم. برای مطالعه اطلاعات فنی دقیقتر در مورد نحوه ساخت برنامه های کاربردی براساس RPC به کتاب Stevens (۱۹۹۹) مراجعه کنید.

## مقدمه ای بر DCE

<sup>528</sup> deferred synchronous RPC

<sup>529</sup> one-way RPC

<sup>530</sup> distributed computing environment

<sup>531</sup> Open Software Foundation

<sup>532</sup> open group

از آنجاییکه DCE جهت اجرا بعنوان لایه انتزاعی بین سیستم های عامل موجود (شبکه) و برنامه های کاربردی توزیعی طراحی شده، می توان آنرا یک سیستم میان افزاری واقعی تلقی نمود. هر چند این سیستم برای UNIX طراحی شده بود، امروزه برای بسیاری از سیستم عامل های مهم از جمله نسخه های ویندوز و VMS و همچنین سیستم های عامل کامپیوترهای شخصی وجود دارد. هدف آن است که خریدار بتواند مجموعه ای از ماشین های موجود را گرفته، نرم افزار DCE را به آن افزوده و سپس برنامه توزیعی را اجرا نماید. تمامی این مراحل باستی بدون ایجاد کوچکترین مشکلی برای برنامه های کاربردی موجود (غیرتوزیعی) انجام شود. هرچند قسمت اعظم بسته DCE در فضای کاربر اجرا می شود، در برخی پیکربندی ها باستی یک قطعه (بخشی از سیستم فایل توزیعی) را به هسته اضافه نمود. گروه باز فقط کد منبع را می فروشد و فروشندهان این کد را در سیستم های خود به کار می گیرند.

مدل زیربنایی برنامه نویسی DCE ها، یک مدل مشتری- سرور است که در بخش قبل بررسی شد. فرآیندهای کاربر، جهت دسترسی به سرویس های ازراه دور ایجادی توسط فرآیندهای مشتری، بعنوان مشتری ایفا نمودند. برخی از این سرویس ها بخشی از خود DCE هستند. اما برخی دیگر متعلق به برنامه های کاربردی بوده و توسط سازندگان برنامه های کاربردی نوشته می شوند. تمامی ارتباطات بین مشتری ها و سرورها از طریق RPC ها انجام می شود. برخی از سرویس ها جزو تشکیل دهنده DCE محسوب می شوند. **سرویس فایل توزیعی**<sup>۵۳۳</sup> یک سیستم فایل جهانی است که روش شفاف و واحدی را برای دسترسی به تمام فایل های درون سیستم ایجاد می کند. این سرویس یا در بر روی سیستم های فایل داخلی میزبان ایجاد می شود و یا به جای آنها مورد استفاده قرار می گیرد. سرویس فهرست<sup>۵۳۴</sup> جهت ردیابی محل تمامی منابع موجود در سیستم بکار برده می شود. این منابع شامل ماشین ها، چاپگرهای سرورها، داده ها و سایر مواردی است که ممکن است هریک از نظر جغرافیایی در گوشه ای از جهان قرار داشته باشند. سرویس فهرست به فرآیند امکان می دهد تا بدون توجه به محل استقرار منبع، آنرا جستجو نماید مگر آنکه خود فرآیند تمایل به دانستن مکان منبع داشته باشد. **سرویس امنیتی**<sup>۵۳۵</sup> امکان محافظت از تمامی انواع منابع را فراهم می آورد تا فقط افراد تأیید شده و مجاز امکان دسترسی داشته باشند. و نهایتاً **سرویس زمان توزیعی**<sup>۵۳۶</sup> موجب همگامی جهانی تمامی ساعتهاي ماشین های مختلف می شود. همانطور که در فصل بعد خواهیم گفت، اطلاع از زمان جهانی باعث هماهنگی سیستم های توزیعی خواهد شد.

## Aهداف DCE RPC

اهداف سیستم DCE PRC تا حدودی مانند مواد قبل است. اول و مهمتر از همه اینکه سیستم PRC به مشتری امکان می دهد تا صرفاً با فراخوانی یک روال محلی به سرویس از راه دور موردنظر دسترسی پیدا کند. این رابط باعث می شود تا برنامه های مشتری (

<sup>533</sup> distributed file service

<sup>534</sup> directory service

<sup>535</sup> distributed service

<sup>536</sup> distributed time service

يعني برنامه هاي کاربردي) به روشی ساده و آشنا برای اغلب برنامه نویسان نوشته شود. همچنین موجب تسهيل اجرای حجمهاي بالايی از کد موجود در يك محيط توزيعي بدون نياز به اعمال تغييرات زياد (يا حتی بدون هيج تغييري) مي شود.

سيستم PRC وظيفه دارد تا تمامي جزئيات را از مشتریان و تا حدودي از سرورها هم مخفی نگه دارد. بدین منظور، سیستم PRC می تواند بطور اتوماتیک سرور درست را مکان یابی نموده و سپس ارتباط بین نرم افزار<sup>۵۳۷</sup> مشتری و سرور را راه اندازی کند (این ويژگي غالباً متصل شدن<sup>۵۳۸</sup> ناميده مي شود). همچنین می تواند کار تبادل پیام را در هردو طرف هدایت کرده و در صورت نیاز (مانند موادردي که پارامترها آرایه بزرگی را تشکيل مي دهد) آنها را تجزие و مجدداً سرهم بندی می کند. در نهایت، سیستم PRC می تواند بطور اتوماتیک کار تبدیلات نوع داده بین مشتری و سرور ، حتی در موادرديکه روی معماري هاي مختلف اجرا و ترتيب بايت مختلفي داشته باشد، را انجام دهد.

در اثر قابلیت سیستم PRC در مخفی سازی جزئیات، میزبان ها و مشتری ها به طور قابل توجهی مستقل از یکدیگر هستند؛ بطوریکه می توان مشتری را به یک زبان مثل Java و سرور را به زبان دیگري از قبیل C نوشت و بالعکس. سرور و مشتری می توانند روی پایگاه های سخت افزاری مختلف اجرا و از سیستم های عامل مختلف استفاده نمایند. بعلاوه، بدون هیچگونه دخالتی از جانب مشتری یا سرور، از انواع پروتکل های شبکه و نمایش های داده پشتیبانی نمود.

### نوشتن مشتری و سرور

سیستم DCE RPC از اجزای مختلفی از قبیل زبان، کتابخانه، برنامه های محافظه<sup>۵۳۹</sup> ، برنامه های کمکی و بسیاری دیگر تشکیل شده است. این اجزاء به کمک هم امکان نوشتن مشتری و سرور را فراهم می کنند. در این بخش، هر یک از اجزاء و نحوه ارتباط آنها با یکدیگر را تشریح خواهیم کرد. کل فرآیند نوشتن و استفاده از مشتری و سرور PRC در شکل ۴-۱۲ خلاصه شده است.

شکل ۴-۱۲ - مراحل نوشتن مشتری و سرور در DCE RPC (داخل شکل: رابط تعريف فایل، کمپایلر، کد مشتری، مشتری مجازی، سرپیام، سرور مجازی، کد سرور، کمپایلر C، کمپایلر C، فایل object مشتری، فایل object مشتری مجازی، فایل object سرور مجازی، فایل object سرور، پیوند دهنده، کتابخانه زمان اجرا، کتابخانه زمان اجرا، پیوند دهنده، پاینده مشتری، پاینده سرور).

در يك سیستم مشتری- سرور، عاملی به نام زبان تعريف رابط یا IDL باعث اتصال تمام اجزاء می شود . این چسب قوی امکان اعلان روال<sup>۵۴۰</sup> را به صورتی بسیار مشابه نمونه های اولیه توابع<sup>۵۴۱</sup> ANSI C فراهم می آورد. بعلاوه، فایل های IDL ممکن است حاوی تعاریف نوع<sup>۵۴۲</sup>، اعلان های ثابت<sup>۵۴۳</sup> و دیگر اطلاعات لازم جهت مرتب سازی صحیح

<sup>۵۳۷</sup> binding

<sup>۵۳۸</sup> daemon

<sup>۵۳۹</sup> procedure declaration

<sup>۵۴۰</sup> function prototypes

<sup>۵۴۱</sup> type definitions

<sup>۵۴۲</sup> constant declaration

پارامترها و استخراج نتایج<sup>۵۴۳</sup> باشند. در حالت ایده آل، تعریف رابط بایستی حاوی تعریفی رسمی از وظیفه روال‌ها هم باشد. اما چنین تعریفی در حال حاضر بالاتر از دانش و توان موجود بوده و به همین دلیل، در تعریف رابط فقط نحوه ادای فراخوانی‌ها ارائه می‌شود نه معنای آنها. در بهترین حالت، نویسنده می‌تواند با افزودن چند کامنت<sup>۵۴۴</sup> عملکرد روال‌ها را تشریح نماید.

یکی از المان‌های بسیار مهم هر فایل IDL، وجود شناسه‌ای کاملً منحصر به فرد برای رابط مورد نظر است. این شناسه توسط مشتری در اولین پیام RPC ارسال شده و سرور صحت آنرا تأیید می‌نماید. بدین ترتیب، چنانچه مشتری سهواً به سرور اشتباه یا حتی نسخه قدیمی تری از سرور درست وصل شود، سرور خطا را تشخیص داده و مانع از اتصال خواهد شد.

تعاریف رابط و شناسه‌های منحصر به فرد در DCE ارتباط نزدیکی با هم دارند. همانطور که در شکل ۴-۱۲ به نمایش درآمده، غالباً اولین قدم در نوشتن برنامه کاربردی مشتری/سرور، فراخوانی برنامه *uuidgen* و درخواست از آن برای تولید پیش‌نمونه فایل IDL حاوی شناسه رابط است. این شناسه بایستی در بین تمامی رابط‌های قبلی تولیدشده بوسیله *uuidgen* منحصر به فرد باشد. این یگانگی با کد کردن محل و زمان ایجاد در داخل آن تضمین می‌شود. کد مذکور شامل یک عدد دو دویی ۱۲۸ بیتی است که در فایل IDL به صورت رشته ASCII مبنای شانزده نمایش داده می‌شود.

مرحله بعد، ویرایش فایل IDL و قراردادن اسمی روال‌های از راه دور و پارامترهای آنها در فایل است. باید بدانیم که RPC کاملً شفاف نیست- بعنوان مثال، مشتری و سرور نمی‌توانند متغیرهای جهانی مشترکی داشته باشند - اما قوانین IDL مانع از بیان ساختارهای فاقد پشتیبانی می‌شود.

پس از تکمیل فایل IDL، کمپایلر IDL جهت پردازش فراخوانی می‌شود. در خروجی کمپایلر IDL سه فایل به چشم می‌خورد:

۱. فایل سرپیام ( وعنوان مثال، *interface.h* به زبان C)

۲. مشتری مجازی<sup>۵۴۵</sup>

۳. سرور مجازی<sup>۵۴۶</sup>

فایل سرپیام حاوی شناسه منحصر به فرد، تعاریف نوع، تعاریف ثابت‌ها و پیش‌نمونه‌های تابع است که بایستی (به کمک `#include`) در هر دو کد سرور و مشتری قرار داده شوند. مشتری مجازی حاوی رویه‌های واقعی است که توسط برنامه مشتری فراخوانی خواهد شد. این رویه‌ها مسئول جمع آوری و بسته بندی پارامترها به شکل پیام خروجی و سپس فراخوانی سیستم زمان اجرا<sup>۵۴۷</sup> جهت ارسال آن است. مشتری مجازی هم به بازکردن پاسخ و برگرداندن مقادیر به مشتری می‌پردازد. سرور مجازی حاوی روال‌هایی است که پس از رسیدن پیام جدید، بوسیله سیستم زمان اجرا روی ماشین سرور

<sup>543</sup> unmarshal

<sup>544</sup> comment

<sup>545</sup> client stub

<sup>546</sup> server stub

<sup>547</sup> run-time system

فراخوانی می شود. این روال ها متقابلاً روال های واقعی سرور را فراخوانی می کند که مسئول انجام اینکار است. در مرحله بعد، کد سرور و مشتری توسط نویسنده برنامه کاربردی نوشته می شود. سپس ایندو به همراه دو روال مجازی کمپایل می شوند. آنگاه، کد مشتری حاصله و فایلهای object مشتری مجازی بوسیله کتابخانه زمان اجرا به هم متصل و باینری قابل اجرایی را برای مشتری ایجاد می کند. به همین ترتیب، کد سرور و سرور مجازی کمپایل و به هم متصل و باینری سرور را بوجود می آورد. در زمان اجرا، سرور و مشتری راه اندازی می شوند تا برنامه کاربردی هم واقعاً اجرا شود.

### اتصال مشتری به سرور

برای آنکه مشتری بتواند سروری را فراخوانی نماید، سرور بایستی ثبت و آماده پذیرش فراخوانی های ورودی باشد. ثبت سرور امکان ردیابی و اتصال به آنرا توسط مشتری فراهم می آورد. مکان یابی سرور در دو مرحله انجام می شود:

#### ۱. مکان یابی ماشین سرور

۲. مکان یابی سرور (یعنی فرآیند صحیح) روی این ماشین مرحله دوم کمی تخصصی تر از مرحله اول است. به طور خلاصه، لازم است مشتری جهت ایجاد ارتباط با سرور، یک نقطه پایانی<sup>۵۴۸</sup> را در ماشین سرور شناسایی و پیام ها را به آن ارسال نماید. در DCE، جدولی حاوی جفتهای (سرور، نقطه پایانی) بوسیله فرآیندی به نام **DCE**<sup>۵۴۹</sup> در هر ماشین سرور نگهداری می شود. سرور بایستی پیش از ورود درخواستهای جدید، از سیستم عامل تقاضای تعیین نقطه پایانی را نموده و سپس این نقطه به کمک محافظ DCE در جدول نقطه پایانی ثبت و حفظ می شود.

همچنین سرور با فراهم آوردن آدرس ماشین سرور و نامی که بتوان سرور را تحت آن جستجو نمود، در سرویس دایرکتوری ثبت می شود. سپس اتصال مشتری به سرور مشابه شکل ۴-۱۳ انجام می شود.

فرض کنید که مشتری بخواهد به سرور ویدئویی ملحق شود که به صورت محلی تحت نام `/local/multimedia/video/movies` شناخته می شود. مشتری این نام را به سرور دایرکتوری انتقال داده و آن هم آدرس شبکه ماشین اجراکننده سرور ویدئو را باز می گرداند. سپس مشتری به سراغ فرآیند محافظ DCE ماشین (که نقطه پایانی شناخته شده ای دارد) رفته و از آن تقاضا می کند که نقطه پایانی سرور ویدئو را در جدول نقطه پایانی خود جستجو نماید و بازگرداند. با داشتن این اطلاعات می توان RPC را اجرا نمود. در RPC های بعدی دیگر نیازی به این جستجو نیست. بعلاوه، DCE به مشتریان امکان می دهد تا در صورت لزوم، جستجوهای پیشرفته تری را برای یافتن سرور مناسب انجام دهند. همچنین در مواردی که امنیت یا انسجام داده ای اهمیت داشته باشد، از RPC این استفاده می شود.

شكل ۱۴-۱۳- اتصال مشتری به سرور در DCE (داخل شکل: ماشین دایرکتوری، سرور دایرکتوری، جستجوی سرور، ثبت سرویس، ماشین مشتری، مشتری، انجام

<sup>548</sup> end-point

<sup>549</sup> DCE daemon

RPC، تقاضای نقطه پایانی، ماشین سرور، سرور، محافظ DCE، ثبت نقطه پایانی، جدول نقطه پایانی).

### اجرای RPC

RPC واقعی به صورت شفاف و به طور معمول انجام می شود. مشتری مجازی به کمک پروتکل انتخابی در زمان اتصال، پارامترها را برای کتابخانه زمان اجرا مرتب و جهت انتقال آماده می کند. هنگامیکه پیام به طرف سرور می رسد، برحسب نقطه پایانی تعیین شده در پیام ورودی به سمت سرور صحیح مسیردهی می شود. کتابخانه زمان اجرا پیام را به سرور مجازی ردکرده و آنهم پارامترها را مرتب نموده و سرور را فرا می خواند. سپس پاسخ در جهت معکوس بازگردانده می شود.

DCE امکانات معنایی مختلفی را ارائه می کند. در حالت پیش فرض<sup>۵۵۰</sup> یعنی معنای "حداکثر یک مرتبه"<sup>۵۵۱</sup>، هیچ فرآخوانی حتی در صورت خرابی سیستم هم بیش از یک مرتبه انجام نمی شود. این ویژگی در عمل باعث می شود تا چنانچه سرور در حین RPC خراب و سپس به سرعت ترمیم شود، مشتری مجدداً عملیات را تکرار نخواهد کرد زیرا بیم آن می رود که قبل از انجام شده باشد.

در روش دیگر می توان روال مورد نظر را (در فایل IDL) تحت عنوان "احتمال تکرار"<sup>۵۵۲</sup> نشانه گذاری کرد. در اینصورت می توان آنرا بدون هیچ اشکالی چندین مرتبه تکرار نمود. بعنوان مثال، خواندن بلوک مشخصی از یک فایل را می توان تا رسیدن به نتیجه مطلوب بارها تکرار کرد. در صورتیکه RPC احتمال تکرار به دلیل خرابی سرور با شکست مواجه شود، مشتری بایستی تا زمان راه اندازی مجدد سرور صبر کرده و سپس مجدداً تلاش کند. معناهای دیگری هم وجود دارند (اما به ندرت استفاده می شوند) که از آن جمله می توان به انتشار RPC در تمام ماشینهای روی شبکه محلی اشاره کرد. در فصل ۸ در مبحث مربوط به RPC به هنگام شکست<sup>۵۵۳</sup> مجدداً در مورد این معناها برای RPC صحبت خواهیم کرد.

### ۴-۳- ارتباطات پیام گرا

فرآخوانی های روال از راه دور<sup>۵۵۴</sup> و احضار شیء از راه دور<sup>۵۵۵</sup> در مخفی سازی ارتباطات در سیستم های توزیعی نقش دارند؛ به این معنی که باعث ارتقاء شفافیت دسترسی می شوند. متأسفانه هیچ سیستمی برای همه موارد مناسب نیست. در برخی موارد که نمی توان اطمینان داشت که طرف دریافت کننده در زمان صدور درخواست در حال اجراست، لزوم وجود سرویس های جایگزین بیش از پیش آشکار می شود. به همین ترتیب، ماهیت ذاتاً همگام RPC ها - که باعث بلوکه شدن مشتری تا زمان پردازش درخواست او می شود- لازم است گاهی با چیز دیگری جایگزین شود.

<sup>550</sup> default

<sup>551</sup> at-most-once

<sup>552</sup> idempotent

<sup>553</sup> failure

<sup>554</sup> remote procedure calls

<sup>555</sup> remote object invocations

این چیز دیگر همان پیام دهی است. در این بخش، روی ارتباطات پیام گرا در سیستم های توزیعی تمرکز کرده و به همین منظور ابتدا نگاه دقیق تری به مفهوم رفتار همگام سازی و اثرات جانبی آن خواهیم داشت. سپس راجع به سیستم های پیام دهی بحث خواهیم کرد که در آنها فرض بر این است که طرفین در زمان ارتباط در حال اجرا نیز هستند. در پایان، انواعی از سیستم های صفتی بندی پیام را مطالعه می کنیم که به فرآیندها امکان می دهد تا حتی در صورتی هم که طرف مقابله در زمان شروع ارتباط در حال اجرا نباشد، اطلاعات را تبادل نمایند.

#### ۱-۴-۳-۱ ارتباطات ناپایدار پیام گرا

بسیاری از سیستمهای توزیعی و برنامه های کاربردی مستقیماً بر روی مدل ساده پیام گرای ارائه شده توسط لایه انتقال ساخته شده اند. جهت درک بهتر مفهوم و اهمیت سیستم های پیام گرا بعنوان بخشی از راه حلهای میان افزاری، ابتدا راجع به پیام دهی از طریق سوکتهای لایه انتقال بحث خواهیم کرد.

#### سوکتهای برکلی

مسئله استانداردسازی رابط لایه انتقال بسیار مورد توجه قرار گرفته تا برنامه نویسان بتوانند به کمک مجموعه ای از عملیات اولیه<sup>۵۶</sup> ساده، از کل مجموعه پروتکلهای پیام دهی استفاده نمایند. بعلاوه، استفاده از رابطهای استاندارد موجب تسهیل انتقال برنامه کاربردی مورد نظر به ماشین دیگر می شود. به عنوان نمونه، مختصراً راجع به **رابط سوکتها**<sup>۵۷</sup> به همان صورتی که در سالهای ۱۹۷۰ در UNIX برکلی ارائه شد، بحث می کنیم. یکی دیگر از رابطهای مهم XTI<sup>۵۸</sup> یا رابط انتقالی X باز<sup>۵۹</sup> است که قبلًا به نام رابط لایه انتقال (TLI) خوانده و توسط AT&T ابداع شده است. سوکت ها و XTI از نظر مدل برنامه نویسی شبکه ای بسیار مشابه، ولی از نظر مجموعه عملیات اولیه متفاوت هستند. از نظر مفهومی، سوکت در واقع نقطه پایانی ارتباط بوده و برنامه کاربردی می تواند داده هایی را که باید به شبکه زیرین ارسال شود، در آن بنویسد و داده های ورودی را از آن بخواند. سوکت انتزاعی از ارتباط واقعی برروی نقطه پایانی است که توسط سیستم عامل محلی برای یک پروتکل انتقال خاص مورد استفاده قرار می گیرد. ذیلًا روی عملیات اولیه سوکت TCP تمرکز می کنیم که در شکل ۴-۱۴ ارائه شده است.

سرورها در غالب موارد چهار عمل اولیه را، به همان ترتیب داده شده، اجرا می کنند. وقتی عمل اولیه سوکت را خوانده می شود، فرآخواننده به ازای یک پروتکل انتقال خاص، یک نقطه پایانی ارتباطی جدید هم ایجاد می کند. از نظر داخلی، ایجاد چنین نقطه پایانی ارتباطی به این معناست که سیستم عامل محلی منابع را جهت فضادادن به پیامهای ارسالی و دریافتی برای آن پروتکل خاص رزرو می کند.

عمل اولیه *bind* یک آدرس محلی را به سوکت جدید احداث مربوط می کند. بعنوان مثال، سرور با یستی آدرس IP ماشین خود را به همراه

<sup>556</sup> primitive

<sup>557</sup> sockets interface

<sup>558</sup> X/ open Transport Interface

یک شماره درگاه (احتمالاً مشهور) به سوکت وصل کند. وصل کردن برای سیستم عامل به این معناست که سرور قصد دارد پیام ها را فقط روی آدرس و درگاه مورد نظر دریافت کند. عمل اولیه *listen* فقط در ارتباطات اتصال گرا فرآخوانی می شود. این فرآخوانی غیربلوکه است و به سیستم عامل محلی امکان می دهد تا با فرکافی را برای حداقل تعداد مورد نظر اتصالاتی که فرآخواننده قصد پذیرش دارد، رزرو کند.

عمل اولیه	معنی
Bind	آدرس محلی جدید را به یک سوکت ضمیمه کن
Listen	تمایل خود را برای پذیرش اتصالات جدید اعلام کن
Accept	تا زمان رسیدن درخواست اتصال جدید، فرآخواننده را بلوکه کن
Connect	بطور به صورتی پویا برای ایجاد اتصال تلاش کن
Send	برخی داده ها را روی اتصال ارسال کن
Receive	برخی داده ها را روی اتصال دریافت کن
Close	اتصال را آزاد کن

شکل ۴-۱۴-عملیات اولیه سوکت برای TCP/IP

فرآخوانی عمل اولیه *accept* باعث بلوکه شدن فرآخواننده تا زمان رسیدن درخواست اتصال می شود. پس از رسیدن درخواست، سیستم عامل محلی سوکت جدیدی با همان ویژگی های سوکت اصلی ایجاد کرده و آنرا به فرآخواننده عوتد می دهد. این روش به سرور امکان می دهد تا بعنوان مثال، از طریق ایجاد اتصال جدید، فرآیندی را ایجاد کند که بعد از ارتباط واقعی را هدایت خواهد کرد. در این حین، سرور می تواند به عقب بازگردد و منتظر درخواست اتصال دیگر روی سوکت اصلی باقی بماند.

حال بیایید نگاهی به طرف مشتری بیندازیم. در اینجا هم ابتدا باید با استفاده از عمل اولیه سوکت، یک سوکت ایجاد شود؛ اما از آنجاییکه سیستم عامل می تواند پس از ایجاد اتصال، بصورتی پویا درگاه را تخصیص دهد، دیگر نیازی به اتصال آشکار سوکت به آدرس محلی نیست. عمل اولیه *connect* باعث تعیین آدرسی از لایه انتقال توسط فرآخواننده می شود که درخواست اتصال باید به آن ارسال شود. مشتری تا زمان ایجاد موقتیت آمیز اتصال، بلوکه شده و پس از آن طرفین می توانند به کمک عملیات اولیه *connect* و *receive* اقدام به تبادل اطلاعات نمایند. در نهایت اینکه بستن(پایان) اتصال در حین استفاده از سوکتها متقارن بوده و با فرآخوانی عمل اولیه *close* توسط هر دو طرف مشتری و سرور امکان پذیر می شود. الگوی کلی مشتری و سرور در ارتباطات اتصال گرا با استفاده از سوکتها در شکل ۴-۱۵ نمایش داده شده است. جزئیات

مربوط به برنامه ریزی شبکه‌ها با استفاده از سوکت‌ها و دیگر رابط‌ها در محیط UNIX را در کتاب (۱۹۹۸) Stevens مطالعه نمایید.

### رابط رد کردن پیام (MPI)

همگام با ظهور مولتی کامپیوتراهاي پربازد، سازندگان در پي ارائه عملیات اولیه پیام گرایی هستند که بتوانند به کمک آنها برنامه های کاربردی بسیار بهره ور را به راحتی بنویسند. به این معنا که عملیات اولیه باید (جهت تسهیل ایجاد برنامه کاربردی) در سطح انتزاع قابل قبولی بوده و پیاده سازی آنها سربار حداقلی ایجادکنند. سوکت‌ها به دو دلیل فاقد کارآیی تصور می‌شوند. اولاً، به دلیل پشتیبانی از عمل‌های اولیه ساده *send* و *receive*، از نظر انتزاع در حد نادرستی قراردادشند. ثانیاً، هدف از طراحی سوکتها ایجاد ارتباط در سرتاسر شبکه‌ها با استفاده از مجموعه‌های پروتکلی عام منظوره ای از قبیل TCP/IP بود. به همین دلیل برای پروتکل‌های اختصاصی طرح ریزی شده برای شبکه‌های میان ارتباطی پرسرعت، از قبیل آنها یکی که در خوش‌های سرور<sup>۵۶۹</sup> با کارآیی بالا مورد استفاده قرار می‌گیرند، مناسب نمی‌باشند. در نتیجه، این نوع پروتکل‌ها نیازمند وجود رابطی هستند که بتوانند با ویژگیهای پیشرفته تری از قبیل اشکال مختلف بافر و همگام سازی عمل کند.

شکل ۴-۱۵-الگوی ارتباطات اتصال گرا با استفاده از سوکتها (داخل شکل: سرور، سوکت، سوکت، سوکت، close، write، accept، listen، bind، نقطه همگام سازی، ارتباطات، سوکت، سوکت، مشتری)

نتیجه این شد که اغلب شبکه‌های میان ارتباطی<sup>۵۶۱</sup> و مولتی کامپیوتراهاي پربازد همراه با کتابخانه‌های ارتباط اختصاصی عرضه می‌شوند. این کتابخانه‌ها گنجینه عظیمی از عمل‌های اولیه ارتباطی سطح بالا و غالباً مؤثر ارائه می‌کردند. البته تمامی کتابخانه‌ها به صورت دوچاره ناسازگار بوده و از این‌رو سازندگان برنامه‌های کاربردی با مشکل قابلیت انتقال<sup>۵۶۲</sup> مواجه شدند.

لزوم عدم وابستگی به سخت افزار و سکوی خاص منجر به تعریف استانداردی برای ردکردن پیام شد که به نام رابط ردکردن پیام<sup>۵۶۳</sup> یا MPI نام گرفت. برای برنامه‌های کاربردی موازی طراحی شده و به همین دلیل جهت ارتباطات ناپایدار سفارشی شده است. همچنین، فرض شده که خرابی‌های جدی از قبیل شکست فرآیند یا بخشهای شبکه حیاتی بوده و به همین دلیل نیازمند احیای اتوماتیک نیستند.

در MPI فرض بر این است که ارتباطات در داخل گروه شناخته شده ای از فرآیندها رخ می‌دهد. به هر گروه یک شناسه نسبت داده می‌شود. همچنین به هر یک از فرآیندهای داخل گروه یک شناسه ( محلی ) تعلق می‌گیرد. بنابراین جفت (ID گروه ، ID فرآیند) منحصرأ یک منبع یا مقصد پیام را شناسایی نموده و به جای آدرس لایه انتقال مورد استفاده قرار می‌گیرد. ممکن است در محاسبه ای، گروههای

<sup>559</sup> server clusters

<sup>560</sup> interconnection networks

<sup>561</sup> portability

<sup>562</sup> Message Passing Interface

فرآیند متعدد و احتمالاً متداخلی موجود باشد که همگی به صورت همزمان در حال اجرا باشند.

در هسته MPI ، عملیات اولیه پیام رسانی جهت پشتیبانی از ارتباطات ناپایدار قرار دارند که مشهودترین آنها در شکل ۴-۱۶ به طور خلاصه ارائه شده است.

ارتباطات ناپایدار همزمان بوسیله عمل های اولیه *MPI-bsend* پشتیبانی می شود. معمولاً هنگامیکه فرستنده پیامی را برای تبادل تسليم می کند، این پیام ابتدا به یک بافر در سیستم زمان اجرای <sup>۵۶۲</sup> MPI کپی می شود. پس از آنکه پیام کپی شد، فرستنده به کار خود ادامه می دهد. سیستم زمان اجرای محلی MPI پیام را از بافر محلی خود جدا کرده و به محض اینکه گیرنده عمل اولیه *receive* را فراخوانی نمود، اقدام به تبادل می کند.

عملیات اولیه	معنی
<i>MPI_bsенд</i>	پیام خروجی را به یک بافر ارسال محلی ضمیمه کن.
<i>MPI_send</i>	پیامی را ارسال و تا زمان کپی شدن به یک بافر یا محلی صبر کن.
<i>MPI_ssенд</i>	پیامی را ارسال و تا زمان آغاز دریافت صبر کن.
<i>MPI_sendrecv</i>	پیامی را ارسال و منتظر جواب بمان.
<i>MPI_isend</i>	منبع را به پیام خروجی انتقال و ادامه بده.
<i>MPI_issend</i>	منبع را به پیام خروجی انتقال و تا زمان آغاز دریافت منتظر بمان.
<i>MPI_recv</i>	پیامی را دریافت کن؛ و صورت عدم وجود پیام آنبلوکه کن.
<i>MPI_irrecv</i>	بررسی کن آیا پیام جدیدی وجود دارد یا نه، اما آنرا بلوکه نکن.

شکل ۴-۱۶- تعدادی از مشهودترین عملیات اولیه ردکردن پیام در MPI

همچنین یک عمل ارسال مسدود به نام *MPI\_send* وجود داردکه هم معنای آن وابسته به پیاده سازی آن است. عملیات اولیه *MPI\_send* می تواند یکی از ایندو کار را انجام دهد؛ **(الف)** مسدود کردن فرآخواننده تا زمانی که پیام در سیستم زمان اجرای MPI در طرف فرستنده کپی می شود و **(ب)** مسدود شدن فرآخواننده تا زمانی که دریافت کننده عملیات دریافت را آغاز کند . ارتباط همگام به گونه ای که فرستنده تا زمانی که درخواستش برای پردازش پذیرفته شود ، مسدود شود ، به توسط عمل اولیه *MPI\_ssенд* تأمین می شود؛ در نهایت قوی ترین شکل ارتباط همگام نیز پشتیبانی می شود؛ یعنی هنگامی که یک فرستنده *MPI\_sendrecv* را فراخوانی می کند، درخواستی به دریافت کننده ارسال کرده و تا زمان بازگشت پاسخ

<sup>۵۶۳</sup> runtime system

مسوده باقی می‌ماند. اساساً، این عمل اولیه متناظر با یک PRC عادی است.

پیامها از بافر کاربر به بافر داخلی سیستم زمان اجرای MPI محلی می‌شوند. این انواع متناظر با شکلی از ارتباط همگام هستند. بوسیله *MPI\_isend*، فرستنده نشانگر به پیام را رد کرده و پس از آن سیستم زمان اجرای MPI اجرای ارتباط را به انجام می‌رساند. فرستنده بی درنگ ادامه می‌دهد. جهت جلوگیری از رونویسی پیام پیش از پایان ارتباط، MPI عملیات اولیه ای را جهت بررسی تکمیل یا حتی بلوکه کردن در صورت لزوم، ارائه می‌دهد. در اینجا هم مانند حالت *MPI\_send*، این که پیام واقعاً به گیرنده منتقل شده باشد یا این که بوسیله سیستم زمان اجرا صرفاً به یک بافرکپی شده باشد، نامشخص باقی می‌ماند.

به همین ترتیب، در *MPI\_issend*، فرستنده فقط نشانگر را به سیستم زمان اجرای MPI رد می‌کند. هنگامی که سیستم زمان اجرا بیان می‌کند که پیام را پردازش نموده فرستنده مطمئن می‌شود که گیرنده پیام را پذیرفته و بر روی آن کار می‌کند.

عملیات *MPI\_recev*، برای دریافت پیام فرآخوانی می‌شود؛ همچنین تا زمان رسیدن پیام، فرآخواننده را مسدود می‌کند. یک نوع ناهمگام هم به نام *MPI irecv* وجود دارد که بوسیله آن گیرنده برای پذیرش پیام اعلان آمادگی می‌کند. گیرنده می‌تواند صحت و سقم واقعی رسیدن پیام را بررسی کرده و یا اینکه تا زمان رسیدن پیام مسدود شود.

معانی عمل‌های اولیه ارتباط MPI همیشه صریح و روشن نیستند، بطوریکه عمل‌های اولیه مختلف را می‌توان بعض‌اً بدون داشتن تأثیر بر صحت برنامه، بجای دیگری استفاده نمود. دلیل واقعی پشتیبانی از تعداد انبوهی از اشکال ارتباطاتی این است که به پیاده سازی‌های سیستم‌های MPI امکانات کافی برای بهینه سازی کارآیی داده شود. در یونان باستان، اعضاي سنا خود را تنها و بالاترین اجماع قابل قبول می‌دانستند، به همین دلیل در همه تصمیم گیری‌ها دخالت می‌کردند. MPI برای برنامه‌های کاربردی موازی با کارآیی بالا طراحی شده است و همین امر باعث سهولت درک تنوع آن در عمل‌های اولیه ارتباطی می‌شود. اطلاعات بیشتر در مورد MPI را در کتاب (1998) Gropp و گروه همکاران مطالعه کنید. مرجع کاملی با بیش از ۱۰۰ کاربرد در MPI با شرح تفصیلی آنها در کتاب (1998a) Snir و گروه همکاران و همچنین (1998) Gropp ذکر شده است.

#### ۵۶۴-۳-۲-۱ ارتباطات پایدار پیام گرا

حال نوبت به بررسی گروه مهمی از سرویسهای میان‌افزار پیام گرا می‌رسدکه غالباً به نام سیستم‌های صفت‌بندی پیام<sup>۵۶۵</sup> یا میان‌افزار پیام گرا<sup>۵۶۶</sup> (MOM) خوانده می‌شود. سیستم‌های صفت‌بندی پیام

<sup>564</sup> Message-Oriented Persistent Communication

<sup>565</sup> message-queuing systems

<sup>566</sup> Message-Oriented Middleware

پشتیبانی قوی از ارتباطات ناهمگام پایدار ارائه می کنند. بطور خلاصه این سیستم ها ظرفیت ذخیره سازی میان مدتی را برای پیام ها ایجاد می کنند ، بدون آنکه حتی لازم باشد فرستنده یا گیرنده حین تبادل پیام فعل باقی بمانند. یکی از موارد مهم تفاوت این نوع سیستم ها با سوکتهاي برکلي و MPI آن است که سیستم هاي صفتندی پیام نوعاً جهت پشتیبانی از تبادلات پیامی مورد استفاده قرار می گیرند که ممکن است به جای چندین ثانیه یا هزارم ثانیه ، چند دقیقه طول بکشند. ابتدا روش کلی سیستم هاي صفتندی پیام را توضیح داده و این بخش را با مقایسه آنها با سیستم هاي متعارف تر، علی الخصوص سیستم هاي پست الکترونیک اینترنت، به پایان می رسانیم.

### مدل صفتندی پیام

ایده اصلی تشکیل دهنده سیستمهای صفتندی پیام آن است که برنامه های کاربردی با درج کردن پیامها در صفحه های خاصی با هم ارتباط برقرار کنند. حتی اگر سرور مقصد در هنگام ارسال پیام بالا نباشد، این پیامها بازهم روی یک سری از سرورهای ارتباطی ارسال شده و نهایتاً به مقصد تحويل داده می شوند . در عمل اکثر سرورهای ارتباطی مستقیماً به یکدیگر متصل هستند . در اصل، هر برنامه کاربردی صفحه خاص خود را دارد و دیگر برنامه های کاربردی می توانند به آن پیام ارسال کنند. هر صفحه فقط بوسیله برنامه کاربردی مربوط به آن قابل خواندن است . اما این امکان هم وجود دارد که برای چندین برنامه کاربردی یک صفحه مشترک وجود داشته باشد.

یکی از جنبه های مهم سیستم هاي صفتندی پیام آن است که به فرستنده فقط تضمین داده می شودکه پیام او نهایتاً در صفتندی درج خواهد شد. هیچ تضمینی در مورد زمان و یا حتی اینکه پیام واقعاً خوانده خواهد شد (کاملاً وابسته به رفتار گیرنده است) ارائه نمی شود.

این معانی ارتباطاتی را امکان پذیر می کنند که از نظر زمانی ارتباط سنتی دارند . بنابراین ، دیگر لزومی ندارد که گیرنده در حین ارسال پیام به صفحه خود در حال اجرا باشد. همچنین ، لزومی ندارد که فرستنده در لحظه ای که پیام او توسط گیرنده گرفته می شود ، فعل باشد. فرستنده و گیرنده می توانند کاملاً به طور مستقل از یکدیگر اجرا شوند . در واقع ، هنگامی که پیامی در صفحه قرار داده می شود ، فارغ از اینکه فرستنده و گیرنده مربوطه در حال اجرا باشد ، تا زمان برداشته شدن در همانجا باقی خواهد ماند . بر همین اساس ، چهار ترکیب مختلف نسبت به حالات اجرای فرستنده و گیرنده بوجود می آید که در شکل ۴-۱۷ دیده می شود.

شکل ۴-۱۷-چهار نوع ترکیب برای ارتباطات دارای پیوند سنت در استفاده از صفحه ها (داخل شکل:الف)فرستنده در حال اجرا-گیرنده در حال اجرا-ب)فرستنده در حال اجرا-گیرنده غیر فعل-ج)فرستنده غیر فعل-گیرنده در حال اجرا-د)فرستنده غیر فعل-گیرنده غیرفعل).

در شکل ۴-۱۷ (الف) ، فرستنده و گیرنده در کل زمان مخابره پیام در حال اجرا هستند.در شکل ۴-۱۷ (ب)، فقط فرستنده در حال اجراست و گیرنده غیر فعل، به این معنی که در این حالت تحويل پیام غیر ممکن است. با این وجود، فرستنده بازهم قادر به ارسال پیامها می باشد. ترکیب فرستنده غیر فعل و گیرنده در حال اجرا

در شکل ۴-۱۷ (ج) نمایش داده شده است. در این حالت، گیرنده قادر به خواندن پیام های ارسالی برای خود است، اما لزومی ندارد که فرستنده های مربوط به آنها هم در حال اجرا باشند. نهایتاً در شکل ۴-۱۷ (د) مشاهده می کنیم که سیستم، حتی در صورت غیرفعال بودن گیرنده و فرستنده هم، قادر به ذخیره سازی (و احتمالاً تبادل) پیام هاست.

اصولاً، پیام ها می توانند حاوی هر نوع داده ای باشند. تنها نکته مهم از بعد میان افزاری، آدرس دهی مناسب پیام ها است. در عمل، آدرس دهی از طریق ایجاد یک نام منحصر به فرد در تمامی سیستم برای صف مقصد انجام می شود. ممکن است در بعضی موارد، برروی اندازه پیام محدودیت وجود داشته باشد؛ گرچه سیستم زیرین می تواند عمل تجزیه و سرهم بندي مجدد پیام های بزرگ را به گونه ای شفاف (مخفي) برای برنامه های کاربردي انجام دهد. يكی از اثرات این روش آن است که رابط اصلی ارائه شده برای برنامه های کاربردي می تواند مشابه شکل ۴-۱۸ بسیار ساده شود.

عملیات اولیه	معنی
Put	پیام را به صف مورد نظر ضمیمه کن
Get	تا زمانیکه صف مورد نظر خالی باشد، مسدود شو و پیام اول را بردار
Poll	پیامهای صف مورد نظر را چک کرده و پیام اول را بردار. هرگز مسدود نشو.
Notify	عمل کننده ای نصب کن تا زمانی که پیام درصف مورد نظر قرار داده شود، فرآخوانی شود

شکل ۴-۱۸- رابط اصلی یک صف در یک سیستم صف بندي پیام

عمل های اولیه *Put* بوسیله فرستنده جهت رდکردن پیامی که قرار است به صف خاصی ضمیمه شود، به سیستم زیرین فرآخوانی می شود. همانطور که قبل ام گفتیم، این فرآخوانی غیر انسدادی است. عمل اولیه *Get* فرآخوانی انسدادی است که یک فرآیند تأیید شده می تواند بوسیله آن پیامی را که بیشترین مدت در صف بوده را بردارد. فرآیند فقط در صورت خالی بودن صف قابل انسداد است. انواع مختلف این فرآخوانی امکان جستجوی پیامی خاص رادر صف، مثلاً براساس اولویت یا هماهنگی با یک الگو امکانپذیر می کند. گونه غیر انسدادی بوسیله عمل اولیه *Poll* ارائه می شود. در صورت خالی بودن صف یا چنانچه نتوان پیام خاصی را پیدا کرد، فرآیند فرآخوانی به کار خود ادامه می دهد.

در نهایت اینکه، اغلب سیستم های صف بندي به فرآیند امکان می دهد تا عمل کننده ای<sup>۵۶۷</sup> به نام کارکرد بازفرآخوانی<sup>۵۶۸</sup> ایجاد کند تا هر زمانی که پیام در صف قرار داده می شود، احضار شود. از بازفرآخوانی ها همچنین می توان جهت آغاز اتوماتیک فرآیندی برای برگیری پیام ها از صف در زمانی استفاده نمود که فرآیند دیگری فعال نباشد. این روش غالباً بوسیله یک برنامه محافظت در طرف گیرنده پیاده سازی می شود. این محافظت دائماً صف را از نظر

<sup>567</sup> handler

<sup>568</sup> callback function

وجود پیامهای جدیدالورود بررسی و به نحو مناسب آنها را اداره می کند.

### معماری عمومی سیستم های صفتی بندی پیام

احازه دهید نگاه دقیقتری به ظاهر کلی یک سیستم صفتی بندی پیام داشته باشیم. یکی از اولین محدودیت‌ها این است که پیام‌ها را می‌توان فقط در صفحه‌هایی قرار داد که محلی فرستنده باشند؛ یعنی صفحه‌ای همان ماشین یا حداقل روی ماشینی نزدیک و برروی همان LAN، بطوریکه به کمک یک RPC به نحو مناسب قابل دسترسی باشد. چنین صفحه‌بندی مبنی<sup>۵۶۹</sup> خوانده می‌شود. به همین ترتیب، پیامها فقط از طریق صفحه‌های محلی قابل خواندن هستند. با این وجود، پیامی که در صفحه قرار می‌گیرد حاوی توصیف صفحه مقصودی<sup>۵۷۰</sup> است که باید به آن انتقال داده شود. از وظایف سیستم صفتی بندی پیام آن است که صفحه‌هایی برای فرستنده و گیرنده ایجاد کرده و مراقب تبادل پیامها از منبع خود به صفحه مقصود آنها باشد.

لازم است بدانیم که مجموعه صفحه‌ها در سرتاسر ماشین‌های مختلفی توزیع می‌شوند. در نتیجه، برای آنکه یک سیستم صفحه‌بندی پیام قادر به انتقال پیام‌ها باشد، باید نگاشت صفحه‌ها به مکانهای شبکه را در اختیار داشته باشیم. همانطور که در شکل ۴-۱۹ مشاهده می‌کنید، در عمل این بدان معناست که باید حاوی پایگاه داده‌ای (احتمالاً توزیع شده‌ای) از اسمی صفحه<sup>۵۷۱</sup> به مکان‌های شبکه باشد. توجه کنید که چنین نگاشتی<sup>۵۷۲</sup> کاملاً مشابه استفاده از سیستم نام دامنه<sup>۵۷۳</sup> (DNS) در پست الکترونیک اینترنت است. بعنوان مثال، برای ارسال پیام به آدرس منطقی پست steen@cs.vu.nl، سیستم پستی برای یافتن آدرس شبکه‌ای (یا IP) سرور پستی گیرنده، DNS را جستجو خواهد کرد تا پیام انتقال داده شود.

شکل ۴-۱۹-را بسطه بین آدرس دهنده سطح صفحه و آدرس دهنده سطح شبکه (داخلی) فرستنده، لایه صفحه‌بندی، سیستم عامل محلی، جستجوی آدرس سطح انتقال صفحه، آدرس سطح صفحه، پایگاه داده‌ای جستجوی آدرس، گیرنده، لایه صفحه‌بندی، سیستم عامل محلی، شبکه، آدرس سطح انتقال

مدیریت صفحه‌ها بر عهده مدیران صفحه است. غالباً مدیر صفحه با برنامه کاربردی که در حال دریافت و ارسال پیام است، مستقیماً تبادل<sup>۵۷۴</sup> دارد. با این وجود، برخی مدیران صفحه بعنوان مسیر یاب، یا رله<sup>۵۷۵</sup> عمل می‌کنند؛ یعنی پیام‌های ورودی را به دیگر مدیران صفحه انتقال می‌دهند. به این ترتیب، یک سیستم صفحه‌بندی پیام ممکن است تدریجاً تبدیل به یک شبکه فوکانی<sup>۵۷۶</sup> کامل سطح برنامه کاربردی در بالای شبکه کامپیوتري موجود شود. این روش مشابه اولین پیاده سازی MBone روی اینترنت است که در آن فرآیندهای کاربردی به عنوان مسیریاب چندپاره‌ی پیکربندی می‌شوند. همانطور که مشاهده خواهیم کرد، اهمیت چند پاره‌ی از طریق شبکه‌های فوکانی هنوز هم به قوت خود باقی است.

<sup>۵۶۹</sup> source queue

<sup>۵۷۰</sup> destination queue

<sup>۵۷۱</sup> queue names

<sup>۵۷۲</sup> mapping

<sup>۵۷۳</sup> Domain Name System

<sup>۵۷۴</sup> relay

<sup>۵۷۵</sup> overlay network

به چند دلیل رله ها می توانند مفید باشند. بعنوان مثال، در بسیاری از سیستم های صفتی بندی پیام هنوز هیچ سرویس نامگذاری عمومی وجود ندارد که قادر به نگهداری پویایی نگاشت های صفتی به مکان باشد. در حال حاضر، توپولوژی شبکه صفتی بندی ایستا بوده و هر مدیر صفتی باید مستقل<sup>۵۷۶</sup> کپی از نگاشت صفتی به - مکان در اختیار داشته باشد. نگفته پیداست که در سیستم های صفتی بندی مقیاس وسیع، اتخاذ این روش می تواند باعث بروز مشکلات مدیریتی شبکه شود.

یک راه حل برای این مشکل استفاده از تعداد محدودی از مسیریاب هایی است که از توپولوژی شبکه اطلاع داشته باشند. هنگامیکه فرستنده A پیامی را برای مقصد B در صفحه محلی خود قرار می دهد، مطابق شکل ۴-۲۰، این پیام ابتدا به نزدیکترین مسیریاب مثلاً به RI منتقل می شود. سپس، مسیریاب پیام را به سمت B ارسال می کند. به عنوان مثال، RI با بررسی نام B متوجه می شود که پیام باید به مسیریاب R2 انتقال داده شود. فقط هنگامی مسیریاب ها به روزرسانی می شوند که صفتی های اضافه یا حذف شوند و هریک از مدیران صفتی های دیگر باید فقط از محل نزدیکترین مسیریاب اطلاع داشته باشد.

بنابراین رله ها غالباً می توانند به ساخت سیستم های مقیاس پذیر صفتی بندی پیام کمک کنند. با این وجود، همزمان با رشد سیستمهای صفتی بندی پیام، پیکربندی دستی شبکه ها غیرقابل مدیریت خواهد شد. تنها راه حل ممکنه برای مشکل آن است که همانند شبکه های کامپیوتري، از روش های مسیریابي پویا استفاده کنیم. تعجب آور است که چنین روشهايی هنوز در برخی از سیستمهای متداول صفتی بندی پیام نهفته نشده اند.

دلیل دیگر برای استفاده از رله ها این است که پردازش ثانویه پیام ها را امکانپذیر می نماید. بعنوان مثال، ممکن است لازم باشد به دلایل امنیتی یا جهت تحمل خرابی ها، پیامها ثبت شوند. یکی از انواع خاص رله که در بخش بعد مورد بحث قرار خواهیم داد رله ای است که به صورت دروازه عمل می کند، یعنی پیامها را تبدیل به قالب قابل فهم برای گیرنده می کند.

سخن آخر اینکه از رله ها می توان جهت چند پخشی استفاده نمود. در اینصورت، هر پیام ورودی در هر صفحه ارسال قرار داده خواهد شد.

شکل ۴-۲۰-سازمان بندی کلی یک سیستم صفتی بندی پیام با مسیریاب ها (فرستنده A، برنامه کاربردی، دریافت صفحه ارسال صفحه پیام، برنامه کاربردی، برنامه کاربردی، مسیریاب، برنامه کاربردی، گیرنده B).<sup>۵۷۶</sup>

### واسطه های پیام

یکی از موارد مهم کاربرد سیستمهای صفتی بندی پیام، تلفیق برنامه های کاربردی فعلی و جدید به صورت یک سیستم اطلاعات توسعی منسجم و منفرد است. چنین تلفیقی الزام می کند که برنامه های کاربردی قادر به درک پیام های دریافتی باشند. در عمل، این ویژگی باعث میشود تا فرستنده پیام های ارسالی خود را در قالبی همانند قالب گیرنده قرار دهد.

یکی از معایب این روش آن است که در صورتیکه برنامه کاربردی به سیستم افزوده شود که نیازمند قالب پیام جدآگانه ای باشد،

<sup>۵۷۶</sup> message brokers

تمامی گیرنده های احتمالی باید تنظیم شوند تا امکان تولید قالب مورد نظر فراهم شود.

روش دیگری که در پروتکل های متعارف شبکه مورد استفاده قرار می گیرد، توافق بر سر یک قالب مشترک پیام است. متأسفانه، این روش غالباً در مورد سیستم های صفتی بندی پیام صدق نمی کند. مشکل ناشی از سطح انتزاع عمل این سیستم هاست. قالب پیام مشترک فقط در صورتی معنادار و قابل استفاده است که مجموعه فرآیندهایی که از آن قالب استفاده می کنند، واقعاً به اندازه کافی با هم اشتراک داشته باشند. در صورت تفاوت فاحش بین مجموعه برنامه های کاربردی که یک سیستم اطلاعات توزیعی را تشکیل می دهند (که غالباً همینطور هم هست)، بهترین قالب مشترک همان زنجیره بایتهاست.

هرچند تعداد کمی قالب های پیام مشترک برای برخی دامنه های برنامه کاربردی تعریف شده است ، ولی روش کلی و توصیه شده یادگیری نحوه برخورد با قالب های مختلف و ایجاد روشهایی جهت انجام تبدیلات به ساده ترین صورت ممکنه است. در سیستم های صفتی پیام ، این دست تبدیلات بوسیله برخی گره های موجود در شبکه صفتی بندی به نام واسطه های پیام انجام می شود. واسطه های پیام در سیستمهای صفتی بندی پیام بعنوان یک دروازه سطح برنامه کاربردی عمل می کنند. هدف اصلی این واسطه ها، تبدیل پیامهای ورودی به صورتی قابل فهم برای برنامه کاربردی مقصد است. توجه داشته باشید که مطابق شکل ۴-۲۱ ، واسطه پیام در سیستم صفتی بندی پیام صرفاً بعنوان یک برنامه کاربردی دیگر عمل می کند. به بیان دیگر، واسطه پیام غالباً جزء لازم صفتی بندی تلقی نمی شود. شکل: مشتری منبع، واسطه پیام در یک سیستم صفتی بندی پیام (داخل شکل، برنامه واسطه، لایه صفتی بندی، مشتری مقصد، شبکه).

واسطه پیام می تواند برای پیام ها مانند یک مبدل عمل کند. بعنوان مثال، فرض کنید که یک پیام ورودی حاوی جدولی از پایگاه داده ای باشد که در آن رکوردها بوسیله یک حائل پایان رکورد خاص تفکیک شده و حوزه های<sup>۷۷</sup> داخل رکورد طول ثابت و شناخته شده ای دارند. چنانچه برنامه کاربردی مقصد انتظار حائل بین رکورد دیگری را داشته و بعلاوه انتظار داشته باشد که این حوزه ها از طولهای مختلفی برخوردار باشند، از واسطه پیام می توان جهت تبدیل پیامها به قالب مقصد موردنظر استفاده نمود.

در موارد پیشتر تر، واسطه پیام ممکن است به عنوان دروازه سطح برنامه کاربردی عمل نماید که از نمونه های آن می توان به دروازه های مسئول تبدیل بین برنامه کاربردی دو پایگاه داده ای مختلف اشاره کرد. در اینصورت، همواره نمی توان تضمین کرد که تمامی اطلاعات داخل پیام ورودی واقعاً به چیزی مفید برای پیام خروجی تبدیل شوند.

با این وجود، معمولاً از واسطه پیام برای تلفیق برنامه کاربردی شرکتی EAI<sup>۷۸</sup> استفاده می شود که در فصل یک مفصل مورد بحث قرار گرفت. در این حالت، واسطه به جای (صرفاً) تبدیل پیامها، مسئول هماهنگ سازی برنامه های کاربردی بر اساس پیامهای مبادله

<sup>577</sup> fields

<sup>578</sup> enterprise application integration

شده است. در این مدل که انتشار/ اشتراك<sup>۵۷۹</sup> نام دارد، پیام ها توسط برنامه های کاربردی به شکل انتشاری ارسال می شوند. در حالت خاص، این مدل ممکن است پیامی را در مورد موضوع X انتشار داده و سپس به واسطه ارسال کنند. بنابراین، برنامه های کاربردی که نسبت به موضوع X علاقه مند بوده و یا به عبارت دیگر، در آن پیامها مشترک شده اند، این پیام ها را از واسطه دریافت می کنند. اشکال پیشرفتیه دیگری هم از واسطه گری وجود دارد که بحث در مورد آنها را به فصل ۱۳ موكول می کنیم.

در دل واسطه پیام، منبع قوانین و برنامه هایی قرار دارد که می توان به کمک آنها یک پیام از نوع T1 را به شکل T2 تبدیل نمود. مشکل اینکار تعریف قوانین و ایجاد برنامه هاست . اغلب محصولات واسطه پیام با ابزارهای توسعه پیشرفتیه ارائه می شوند، اما در نهایت مجموعه اطلاعات منبع باید از کارشناسان خبره دریافت شود . ذیلاً نمونه ای از محصولات تجاري ذکر می شود که به اشتباه ارائه دهنده "هوشمندی" تلقی می شوند ، اما واقعیت این است که هوش تنها در مغز کارشناسان خبره قرار دارد .

### نکته ای در مورد سیستم های صفحه بندی پیام

با بررسی مجدد آنچه تاکنو ن درمورد سیستم های صفحه بندی پیام گفته شد، به نظر می رسد که این سیستم ها مدتھاست در سرویس های پست الکترونیک پیاده سازی شده و در حال کاربرد هستند. سیستم های پست الکترونیک غالباً از طریق مجموعه ای از سرورهای پستی پیاده سازی می شوند که پیام ها را به نمایندگی از جانب کاربران روی میزبان هایی که مستقیماً به سرور متصل هستند، ذخیره و ارسال می کنند . از آنجایی که سیستم های پست الکترونیک می توانند مستقیماً از سرویس های انتقال زیرین استفاده کنند، مسیریابی کاربرد ندارد. به عنوان مثال، در پروتکل پست اینترنت یا SMPT (Postel، ۱۹۸۲) ارسال پیام از طریق ایجاد اتصال TCP مستقیم به سرور پست مقصد انجام می پذیرد.

وجه تمايز ویژه سیستم های پست الکترونیک از سیستم های صفحه بندی پیام آن است که هدف اصلی سیستم های پست الکترونیک، ایجاد پشتیبانی مستقیم برای کاربران نهایی است. از همین رو می توان بعنوان مثال توضیح داد که چرا برخی از برنامه های کاربردی گروه افزار مستقیماً بر اساس سیستم پست الکترونیک استوار شده اند (Khoshafian and Buckiewicz، ۱۹۹۵). بعلاوه، سیستم های پست الکترونیک از ویژگی های بسیار خاصی از قبیل فیلترینگ خودکار پیام، پشتیبانی از پایگاههای پیشرفتیه داده ای (مثلاً جهت بازیابی آسان پیام های قبل ذخیره شده) و غیره برخوردار هستند.

هدف سیستم های عمومی صفحه بندی پیام فقط حمایت از کاربران نهایی نیست. یک هدف مهم از ایجاد آنها برقراری امکان ارتباطات پایدار در بین فرآیندهاست، مستقل از آنکه آیا فرآیند در حال اجرای برنامه کاربردی کاربر است یا دسترسی به پایگاه داده ای را بر عهده دارد و یا به انجام محاسبات مشغول است . این شیوه باعث می شود تا سیستم های صفحه بندی پیام در قیاس با سیستم های پست الکترونیکی از مجموعه ویژگی های متتنوع تری برخوردار باشند . بعنوان مثال، غالباً نیازی نیست که سیستم های پست الکترونیک

<sup>۵۷۹</sup> publish/ subscribe

برای یک کاربرد عام ، تحویل تضمینی پیام ، اولویت بندی پیام ، امکانات ثبت وقایع ، چند پخشی مؤثر ، تعادل بار ، تحمل خطا و از این قبیل را ارائه دهد .

بنابراین ، سیستم های صفت بندی پیام عام منظوره دارای کاربردهای گسترده ای از قبیل پست الکترونیک ، جریان کار<sup>۵۸۰</sup> ، گروه افزار و پردازش دسته می باشند . با این وجود ، همچنان که قبل از گفتم ، مهمترین زمینه کاربرد آنها تلفیق مجموعه (احتمالاً بسیار پراکنده ای) از پایگاههای داده ای و برنامه های کاربردی به شکل یک سیستم اطلاعاتی است (Hohpe and Woolf ، ۲۰۰۴) . بعنوان مثال ، ممکن است لازم باشد پرس و جویی که در چندین پایگاه داده ای انجام می شود ، به چندین زیر مجموعه تقسیم و به پایگاههای داده ای مختلف ارسال شود . سیستم های صفت بندی پیام ابزار اصلی را برای بسته بندی هریک از زیرپرس وجوها به شکل پیام و مسیردهی آن به پایگاه داده ای مناسب ایجاد می کنند . دیگر امکانات ارتباطی که در این بخش مورد بحث قرار گرفت ، چندان برای این کار مناسب نیستند .

### ۴-۳-۴-مثال: سیستم صفت بندی پیام WebSphere

جهت درک بهتر نحوه عملکرد سیستم های صفت بندی پیام ، اجازه دهید نگاهی به یکی از سیستم های خاص یعنی سیستم صفت بندی پیام بیندازیم که بخشی از محصول IBM WebSphere محسوب می شود . این محصول که سابقاً MQ WebSphere نامیده می شد ، امروزه MQSeries MQ WebSphere خوانده می شود . مستندات فراوانی در ارتباط با وجود دارد ، اما ذیلاً فقط اصول کلی اصلی را ذکر می کنیم . بسیاری از جزئیات مربوط به معماری شبکه های صفت بندی پیام را می توانید در (IBM ۲۰۰۵d ، b ، ۲۰۰۵ ، a) MQ نویسی شبكه های صفت بندی پیام را نمی توان یک شبه یاد گرفت و راهنمای برنامه نویسی MQ (IBM ۲۰۰۵) شاهد خوبی دال بر این مدعاست که از نظریه تا عمل راه فراوان است .

#### شرح کلی

معماری اصلی شبکه صفت بندی MQ ، که نمونه ای از آن را در شکل ۴-۲۲ مشاهده می کنید ، صریح و قابل فهم است . تمامی صفات تحت مدیریت مدیران صفت<sup>۵۸۱</sup> قرار دارند . مدیر صفات مسئول برداشتن پیامها از صفحات ارسال مربوطه و پیش بردن آنها به سمت دیگر مدیران صفات است . بعلاوه ، مدیر صفات مسئول هدایت پیامهای ورودی از طریق برداشت آنها از شبکه زیرین و سپس ذخیره سازی هر پیام در صفحه ورودی مناسب است . جهت درک بهتر مفهوم پیام رسانی لازم است بدانید که حد اکثر سایز پیش فرض برای یک پیام ۴ مگابایت است که تا ۱۰۰ MB هم قابل افزایش می باشد . در یک صفحه معمولاً ۲ GB داده وجود دارد که بسته به سیستم عامل زیرین ، این رقم به راحتی قابل افزایش است .

مدیران صفات به صورت دوتایی از طریق کانالهای پیام<sup>۵۸۲</sup> متصل هستند که انتزاعی از اتصالات سطح انتقال هستند . یک کانال پیام ، اتصال

<sup>580</sup> workflow

<sup>581</sup> queue managers

<sup>582</sup> message channels

مطمئن و یک جهتی بین مدیر صف ارسال و دریافت است که پیامهای صف بندی شده از طریق آن انتقال می یابند. بعنوان مثال، یک کانال پیام بر اساس اینترنت به صورت اتصال TCP اجرا می شود. هر یک از دو سر کانال پیام توسط نماینده کانال پیام<sup>۵۸۳</sup> (MCA) مدیریت می شود. MCA ارسال اصولاً وظیفه ای غیر از بررسی صفات ارسال از نظر وجود پیام، تبدیل آن به شکل بسته سطح انتقال و ارسال آن از طریق اتصال به MCA دریافت مربوطه ندارد. به همین ترتیب، یکی از وظایف اصلی MCA دریافت، گوش دادن برای ورود بسته، باز کردن آن و سپس ذخیره پیام باز شده در صفات مناسب است.

شکل ۴-۲۲- سازمان بندی کلی یک سیستم صف بندی پیام IBM (داخل شکل: مشتری ارسال، برنامه، رابط MQ، مجازی، مسیرهای، مدیریت، مسیرهای، مدیر صفات، صفات ارسال، سرورمجازی، شبکه محلی، صفات دریافت مشتری، مدیر صفات، سرور مجازی، مشتری دریافت، برنامه، مجازی، شبکه شرکتی، ردکردن پیام (ناهمگام)، به سمت دیگر مدیران صفات دور).

مدیران صفات را می توان به همان برنامه کاربردی متصل نمود که صفات را برای آن مدیریت می کنند. در این حالت، صفات ها از برنامه کاربردی به توسط یک رابط استاندارد مخفی می شوند، اما می توان آنها را به نحو مؤثری بوسیله برنامه کاربردی دستکاری نمود. در یک سازمان بندی دیگر، مدیران صفات و برنامه های کاربردی روی ماشین های جداگانه اجرا می شوند. در این حالت، همان رابطی به برنامه کاربردی ارائه می شود که در صورت قرارگرفتن مدیر صفات روی آن ماشین ارائه خواهد شد. در اینصورت، رابط به عنوان میانجی<sup>۵۸۴</sup> اجرا می شود که با استفاده از ارتباط همگام متعارف براساس RPC با مدیر صفات ارتباط پیدا می کند. به این ترتیب، MQ مدلی را که بتوان فقط به صفات های محلی یک برنامه کاربردی دسترسی پیدا کرد، حفظ می کند.

## کانالها

یکی از اجزاء مهم و اصلی MQ، کانالهای پیام است. هر کانال پیام دقیقاً دارای یک صفت ارسال است و پیامهای ارسالی به طرف دیگر از آن انتخاب خواهند شد. انتقال در داخل کانال فقط در صورتی امکانپذیر است که هردو MCA دریافت و ارسال آن برقرار رو در حال اجرا باشند. غیر از راه اندازی دستی MCA ها، روش های متعدد دیگری برای راه اندازی کانال وجود دارد که ذیلاً راجع به آنها بحث خواهیم کرد.

یک روش آن است که برنامه کاربردی با فعال کردن MCA ارسال یا دریافت، سه کانال طرف خود را فعال کند. با این وجود، به دلیل کاهش شفافیت، این روش چندان جالب و مفید نیست. روش بهتر برای آغاز MCA ارسال آن است که صفت ارسال کانال را به صورتی پیکربندی کنیم که در صورتی که پیام برای اولین بار در صفت قرار می گیرد، یک تریگر<sup>۵۸۵</sup> فعال شود. تریگر به عاملی متصل است که MCA ارسال را آغاز می کند تا بتواند پیامها را از صفت ارسال بردارد.

<sup>583</sup> message channel agent

<sup>584</sup> proxy

<sup>585</sup> trigger

روش دیگر، آغاز MCA روی شبکه است. بخصوص ، در صورت فعال بودن یکی از طرفین کanal، طرف مذکور می تواند با ارسال پیام کنترل از طرف دیگر MCA درخواست آغاز نماید. این پیام کنترل به محافظتی ارسال می شود که به آدرس معروفی روی همان ماشین گوش می دهد.

توقف کانالها بطور اتوماتیک و پس از گذشت مدت زمانی معین که هیچ پیام جدیدی در صف ارسال انداخته نشود، انجام می پذیرد. هر MCA دارای مجموعه ای از ویژگی های مربوطه است که رفتار کلی کanal را تعیین می کند. برخی از این ویژگی ها در شکل ۴-۲۳ درج شده است. مقادیر ویژگی MCA ارسال و دریافت باستی سازگار بوده و احتمالاً پیش از برقراری کanal مورد توافق قرار گیرند. بعنوان مثال، هر دو MCA ها باید آشکارا از یک پروتکل انتقال واحد پشتیبانی کنند. یک نمونه از ویژگی های قابل بحث آن است که آیا پیام ها باید به همان ترتیب قرارگیری در صف ارسال، تحویل داده شوند یا خیر. چنانچه یک MCA خواستار تحویل به صورت FIFO باشد، دیگری هم باید با آن هماهنگ باشد. یک نمونه از مقادیر ویژگی قابل توافق ، حد اکثر طول پیام است که حداقل مقدار اعلام شده بوسیله دو MCA انتخاب می شود.

شرح	ویژگی
تعیین کننده پروتکل انتقال مورد استفاده است	نوع انتقال
بیان می کند که پیامها باید به همان ترتیب ارسال تحویل داده شوند	TUOIL FIFO
حد اکثر طول یک پیام	طول پیام
بیانگر حد اکثر دفعات تلاش برای آغاز یک MCA راه دور است	شمارش دفعات تلاش برای راه اندازی
حد اکثر دفعاتیکه MCA تلاش می کند تا پیام دریافتی را در صف قرار دهد	دفعات تکرار تحویل

شکل ۴-۲۳ - برخی از ویژگی های مربوط به نمایندگی های کanal پیام

### انتقال پیام

برای انتقال پیام از یک مدیر صف به مدیر(احتمالاً دور) صف دیگر، لازم است که هر پیام حاوی آدرس مقصد خود باشد که بدین منظور از یک سرپیام استفاده می شود. هر آدرس در MQ شامل دو بخش است. بخش اول شامل نام مدیر صفحی است که پیام باید به آن تحویل داده شود. بخش دوم نام صف مقصد تحت مدیریت آن است که پیام باید به آن ضمیمه شود.

لازم است علاوه بر آدرس مقصد، مسیری هم که پیام باید طی کند مشخص شود. تعیین مسیر از طریق ارائه نام صف ارسال محلی انجام می پذیرد که پیام باید به آن ضمیمه شود. بنابراین لزومی ندارد که مسیر کامل در پیام قید شود. به یاد داشته باشید که هر کanal پیام دقیقاً حاوی یک صف ارسال است. با ذکر اینکه پیام باید به کدام صف ارسال ضمیمه شود، می توان به نحو مؤثری اقدام به تعیین مدیر صفحی نمود که پیام باید به آن ارسال شود.

در اغلب موارد، مسیرها به صورتی واضح در جدول مسیریابی در داخل مدیر صفت ذخیره می شوند. ثبت در جدول مسیریابی به صورت جفتی ( $QM$  مقصد،  $Q$  ارسال) انجام می پذیرد که در آن،  $QM$  مقصد نام مدیرصف مقصد و  $Q$  ارسال نام صف ارسال محلی است که پیام مربوط به مدیر آن صف باید بدان ضمیمه شود. (ورودی به جدول مسیریابی در نام مستعار<sup>۵۸۶</sup> خوانده می شود).

این احتمال وجود دارد که یک پیام پیش از رسیدن به مقصد خود از چندین مدیر صفت رد کند. هر زمانی که پیام تحت این شرایط توسط مدیر صفت میانی دریافت شود، مدیر صرفاً نام مدیرصف مقصد را از عنوان پیام استخراج کرده و در جدول مسیرها به دنبال صف ارسال محلی می گردد که پیام باید به آن ضمیمه شود.

لازم به ذکر است که هر مدیر صف دارای یک نام منحصر به فرد در تمامی سیستم است که به نحو مؤثری بعنوان شناسه آن مدیرصف مورد استفاده قرار می گیرد. مشکل استفاده از چنین اسمی آن است که جایگزینی مدیرصف یا تغییر نام آن بر تمامی برنامه های کاربردی که پیام ها به آن ارسال می شود، تأثیر خواهد گذاشت. می توان با استفاده از یک نام مستعار محلی<sup>۵۸۷</sup> برای اسمی مدیرصف، این دست مشکلات را تا حدودی حل و فصل نمود. نام مستعار تعریف شده داخل مدیرصف  $MI$  نام دیگری برای مدیرصف  $M2$  است، اما فقط برای برنامه های کاربردی رابطه  $MI$  قابل استفاده است.

نام مستعار استفاده از یک نام (منطقی) واحد برای صف را، حتی در صورت تغییر مدیر آن صف هم امکان پذیر می کند. تغییر نام مدیرصف مستلزم تغییر نام مستعار آن در تمام مدیران صف است. اما، کوچکترین تغییری در برنامه های کاربردی ایجاد نخواهد شد.

شكل ۴-۲۴- سازمان بندي کلی یک شبکه صف بندي پیام  $MQ$  با استفاده از جداول و نام های مستعار مسیردهی (داخل شکل: جدول نام مستعار، جدول مسیردهی، جدول نام مستعار، جدول مسیردهی، جدول نام مستعار، جدول مسیردهی، جدول مسیردهی).

اصل کلی استفاده از جداول مسیردهی و نام مستعار در شکل ۴-۲۴ ارائه شده است. بعنوان مثال، می توان با استفاده از نام مستعار محلی  $LAI$ ، برنامه کاربردی متصل به مدیرصف  $QMA$  را به یک مدیرصف دور ارجاع داد. مدیرصف ابتدا در جدول نام مستعار، مقصد واقعی را جستجو خواهد کرد تا مدیرصف  $QMC$  را پیدا کند. مسیر به سمت  $QMC$  در جدول مسیریابی یافته می شود و همین بیان می دارد که پیام های مربوط به  $QMC$  باید به صف خروجی  $SQ1$  ضمیمه شوند که از آن جهت انتقال پیام ها به مدیرصف  $QMB$  استفاده شده است. مدیرصف  $QMB$  هم از جدول مسیریابی جهت ارسال پیام به  $QMC$  استفاده می کند.

اتخاذ این روش جهت تعیین مسیر و نام مستعار منجر به برنامه نویسی رابطی می شود که ساده بوده و به نام رابط صف پیام<sup>۵۸۸</sup> ( $MQI$ ) خوانده می شود. مهمترین عملیات اولیه  $MQI$  در شکل ۴-۲۵ به طور خلاصه ارائه شده اند.

<sup>586</sup> alias

<sup>587</sup> local alias

<sup>588</sup> Message Queuing Interface

عملیات اولیه	شرح
MQopen	یک صف (احتمالاً دور) را باز می کند
MQclose	یک صف را می بندد
MQput	پیامی را در صف باز شده قرار می دهد
MQget	پیامی را از یک صف (محلي) می گيرد

شکل ۴-۲۵-عمل های اولیه موجود در رابط صفت بندی پیام

جهت قراردادن پیامها در صف ، برنامه کاربردی عمل اولیه *MQopen* را فرامی خواند و یک صف مقصد را دریک مدیر صف خاص مشخص می کند . مدیر صف را می توان با استفاده از نام مستعار موجود به صورت محلی نامگذاری کرد . دور یا محلی بودن صف مقصد برای برنامه کاربردی کاملاً مخفی است . همچنین فرآخوانی *MQopen* هنگامی صورت می گیرد که برنامه کاربردی بخواهد پیام ها را از صف محلی خود بگیرد . برای خواندن پیام های ورودی ، فقط صفحهای محلی را می توان باز کرد . پس از دسترسی به صف و پایان برنامه کاربردی ، لازم است با فرآخوانی *MQclose* اقدام به بستن آن نمود .

با استفاده از *MQget* و *MQput* به ترتیب امکان نوشتن و خواندن پیام ها از صف فراهم می شود . در مجموع ، برداشت پیام از صف بر اساس اولویت بندی صورت می پذیرد . پیام های دارای اولویت واحد بر اساس قاعده اولین ورود - اولین خروج برداشته می شوند ، به این معنا که پیامی که طولانی ترین انتظار را داشته ، زودتر از همه خارج می شود . بعلاوه امکان درخواست پیام های خاص وجود دارد . در پایان باید گفت که *MQ* امکاناتی را برای سیگنال دهی به برنامه های کاربردی به محض ورود پیامها ارائه داده و به این ترتیب دیگر لازم نیست که برنامه کاربردی دائماً صف پیام مورد نظر را برای یافتن پیامهای جدید جستجو کند .

### مدیریت شبکه های فوقانی

تا به اینجا می توان نتیجه گرفت که یکی از جنبه های مهم سیستم های مدیریت *MQ* ، اتصال مدیران مختلف صف به یک شبکه فوقانی پایدار است . بعلاوه ، لازم است که این شبکه در طول زمان نگهداری شود . در شبکه های کوچک ، این نگهداری فقط مستلزم اعمال مدیریتی متوسطی است . اما هنگامی که از همین صف بندی پیام جهت تلفیق و تشکیل سیستم های بزرگ موجود استفاده می شود ، اوضاع تا حدودی پیچیده می شود .

یکی از مشکلات مهم *MQ* آن است که شبکه های فوقانی باید به صورت دستی مدیریت شوند . چنین مدیریتی علاوه بر کمال سازی بین مدیران صف ، شامل پرکردن جداول مسیردهی هم می باشد . نگفته پیداست که گسترش این وظیفه بسیار طاقت فرساست . متأسفانه ، پشتیبانی مدیریتی از سیستم های *MQ* فقط به گونه ای پیش می رود که یک مدیر اجرایی هر نوع ویژگی ممکن را قرار داده و هر پیکربندی قابل تصور را اعمال نماید . با این وجود ، لازم است

<sup>589</sup> administrator

کانال ها و جداول مسیردهی بازهم به صورت دستی حفظ و نگهداری شوند.

مؤلفه تابع کنترل کانال<sup>۵۹۰</sup> قلب مدیریت فوچانی است که منطقاً بین نماینده های کانال پیام جای دارد. این مؤلفه به اپراتور امکان می دهد تا دقیقاً برآنچه برسر دو نقطه انتهایی کانال می رود، نظارت کند. علاوه بر ایجاد کانالها و جداول مسیر یابی، این مؤلفه برای مدیریت مدیران صفت استفاده می شود که میزبان نماینده های کانال پیام هستند. از یک نظر، این روش مدیریت فوچانی بسیار شبیه مدیریت سرورهای خوش ای در مواردی است که فقط از یک سرور مدیریت اجرایی استفاده می شود. در این حالت، سرور فقط یک پوسته دور<sup>۵۹۱</sup> برای هریک از ماشین های داخل خوش به همراه چند عملیات جمعی محدود جهت هدایت گروه های ماشین ها ارایه می کند. با این وجود، چنانچه به دنبال حوزه جدیدی برای یافتن راهکار مشکلات جدی و مهم می گردید، کار برروی مدیریت سیستم های توزیعی فرصت بسیار خوبی در اختیاراتان خواهد گذاشت.

#### ۴-۴- ارتباطات جریان گرا

تا به اینجا، مسئله ارتباطات روی تبادل واحدهای اطلاعات کما بیش مستقل و کامل تمرکز یافته بود. از نمونه های آن می توان به درخواست احظار روال، پاسخ به درخواست احضار روال و تبادلات پیام بین برنامه های کاربردی از قبیل سیستم های صفحه زمانی اشاره کرد. ویژگی بارز این نوع ارتباطات آن است که نقطه زمانی انجام ارتباطات کاملاً فاقد اهمیت است. علاوه، سرعت عملکرد سیستم، خواه بسیار بالا و خواه بسیار پایین، کوچکترین تأثیری بر صحت کار نخواهد داشت.

اما در برخی انواع ارتباطات مسئله زمان نقش کلیدی ایفا می کند. بعنوان مثال تصور کنید که یک جریان صوتی به صورت رشته ای از نمونه های ۱۶ بیتی ساخته شده باشد که هریک بیانگر دامنه موج صوت بر حسب مدولاسیون کد پالس<sup>۵۹۲</sup> (PCM) باشد. همچنین تصور کنید که این جریان صوتی بیانگر کیفیت یک CD است، به این معنا که موج صوتی اصلی تحت فرکانس<sup>Hz</sup> ۴۴۱۰۰ نمونه گیری شده است. برای تولید صوت اصلی، لازم است که نمونه های داخل جریان صوتی به همان ترتیب قرارگیری در جریان اجرا شوند، اما فوائل زمانی دقیقاً ۱/۴۴۱۰ ثانیه ای هم در نظر گرفته شود. اجرا تحت سرعتهای مختلف، نسخه اشتباهی از صوت اصلی ایجاد خواهد کرد.

مسئله مورد بحث در این بخش آن است که یک سیستم توزیعی باید چه امکاناتی جهت تبادل اطلاعات وابسته به زمان از قبیل جریانات صوتی و شکلی ارائه دهد. پروتکلهای شبکه ای مختلف در ارتباط با ارتباطات جریان گرا در کتاب<sup>۵۹۳</sup> (Halsall، ۲۰۰۱)، Steinmetz and Nahrstedt (۲۰۰۴) تعریف کلی از مسائل چندرسانه ای ها مطرح کرده که بخشی از آن در رابطه با ارتباط جریان گرا است.

<sup>۵۹۰</sup> channel control function

<sup>۵۹۱</sup> remote shell

<sup>۵۹۲</sup> stream-oriented communication

<sup>۵۹۳</sup> Pulse Code Modulation

پردازش پرس وجو روی جریانات داده ای در کتاب (Babcock ، ۲۰۰۲) و گروه همکاران مورد بحث قرار گرفته است.

#### ۴-۴-۱-پشتیبانی از رسانه های پیوسته

پشتیبانی از تبادل اطلاعات وابسته به زمان غالباً در قالب پشتیبانی از رسانه های پیوسته مطرح می شود. منظور از رسانه ابزارهایی است که جهت انتقال اطلاعات مورد استفاده قرار می گیرد. این ابزارها شامل رسانه ذخیره سازی و ارسال، رسانه ارائه از قبیل مانیتور و غیره است. یکی از مهم ترین انواع رسانه ها نحوه ارائه این اطلاعات یا به بیان دیگر، نحوه کدی کردن اطلاعات در یک سیستم کامپیوتري است. برای انواع مختلف اطلاعات از روش‌های ارائه مختلفی استفاده می شود. به عنوان مثال، متن معمولاً به صورت ASCII یا یونی کد<sup>۵۹۴</sup> و تصاویر در قالبهای مختلفی از جمله GIF یا JPEG کد می شوند. جریانات صوتی را می توان در یک سیستم کامپیوتري مثلاً با استفاده از PCM به صورت نمونه های<sup>۱۶</sup> بیتی تبدیل کرد.

در رسانه های (ارائه) پیوسته<sup>۵۹۵</sup>، روابط زمانی بین اقلام مختلف داده ای در تفسیر درست معنا ومفهوم واقعی داده اهمیت اساسی دارد. مثال مربوط به تولید موج صوتی بوسیله اجرای جریان صوتی را به خاطر بیاورید. بعنوان نمونه دیگر، مسئله حرکت را ذکر می کنیم. حرکت را می توان بوسیله مجموعه ای از تصاویری القاء کرد که باید در فواصل زمانی  $T$  یکنواخت (ممولاً ۳۰-۴۰ میلی ثانیه برای هر شکل) نمایش داده شوند. نمایش صحیح علاوه بر نمایش عکس ها در ترتیب درست، مستلزم حفظ فرکانس ثابت  $I/T$  شکل در هر ثانیه است.

در رسانه های (ارائه) گسته<sup>۵۹۶</sup>، برخلاف نوع پیوسته، روابط زمانی بین اقلام داده ای اهمیت اساسی را در تفسیر درست داده ها ندارند. از نمونه رسانه های گسته می توان به نمایش متن و تصاویر ثابت و همچنین فایلهای کد شیء و کد قابل اجرا اشاره کرد.

#### جریان داده ای

غالباً سیستم های توزیعی جهت تبادل اطلاعات وابسته به زمان از جریانات داده ای<sup>۵۹۷</sup> پشتیبانی می کنند. جریان دادهای در واقع رشته ای از واحدهای داده است. جریانات داده ای را می توان در هردو نوع رسانه های پیوسته و گسته مورد استفاده قرار داد. از نمونه های جریانات داده ای گسته (بایت گرا) می توان به لوله های UNIX<sup>۵۹۸</sup> یا اتصالات TCP/IP اشاره کرد. اجرای فایلهای صوتی غالباً نیازمند ایجاد جریان داده ای پیوسته بین فایل و وسیله صوتی است.

مسئله زمانبندی در جریانات دادهای پیوسته نقش کلیدی ایفا می کند. برای داشتن ویژگی های زمانی، معمولاً موضوع تفاوت بین روش‌های

<sup>594</sup> unicode

<sup>595</sup> continuous ( representation) media

<sup>596</sup> discrete ( representation) media

<sup>597</sup> data streams

<sup>598</sup> UNIX pipes

مختلف انتقال مطرح می شود. در روش انتقال ناهمگام<sup>۵۹۹</sup>، اقلام داده جریان یکی پس از دیگری ارسال می شوند؛ اما محدودیت زمانی دیگر وجود ندارد. این ویژگی غالباً در جریانات داده ای گستره هم صدق می کند. بعنوان مثال، یک فایل را می توان به صورت جریان داده ای ارسال کرد. اما این که چه زمانی هر قلم به پایان می رسد، قادر اهمیت است.

در روش انتقال همگام<sup>۶۰۰</sup>، به ازای هر واحد در جریان داده یک حد اکثر تأخیر انتها به انتهای<sup>۶۰۱</sup> تعریف می شود. انتقال واحد داده با سرعتی بسیار بیشتر از حد اکثر تأخیر تحمل شده قادر اهمیت است. بعنوان مثال، یک حسگر درجه حرارت تحت سرعت مشخصی نمونه گیری کرده و از طریق شبکه به اپراتور انتقال می دهد. در این حالت، ممکن است لازم باشد که زمان انتشار انتها به انتهای در شبکه حتماً کمتر از فاصله زمانی بین برداشت نمونه ها باشد، اما انتشار نمونه ها تحت سرعتی بسیار بیشتر از سرعت لازم هم مشکل زا نخواهد بود.

در روش تبادل تک زمانی<sup>۶۰۲</sup>، واحدهای داده ای باید درست سروقت ارسال شوند. به این معنی که تبادل داده در معرض نوعی حد اکثر و حداقل تأخیر انتها به انتها قرار دارد که به نام لرزش (تأخر) محدود شده معروف است. از آنجاییکه روش تبادل تک زمانی نقش کلیدی در ارائه شکل و صوت ایفا می کند، خصوصاً در سیستم های توزیعی چندرسانه ای اهمیت ویژه ای دارد. در این فصل، فقط روی جریانات داده ای تک زمانی با استفاده از مخابره تک زمانی تمرکز کرده و به همین دلیل از این پس فقط جریانات نامیده می شود.

جریانات ممکن است ساده یا پیچیده باشند. جریان ساده<sup>۶۰۳</sup> متشکل از فقط یک زنجیره داده است، درحالیکه جریان پیچیده<sup>۶۰۴</sup> متشکل از چندین جریان ساده مرتبط به نام زیرجریان<sup>۶۰۵</sup> است. ارتباط بین زیرجریانات در یک جریان پیچیده غالباً وابسته به زمان نیز می باشد. بعنوان مثال، صوت استریو را می توان بواسیله یک جریان پیچیده متشکل از دو زیر جریانی که دائماً همگام می شوند، انتقال داد. به بیان دیگر، واحدهای داده هر جریان را باید به صورت جفت در ارتباط انتقال داد تا تأثیر استریو حفظ شود. نمونه دیگری از جریانات پیچیده انتقال فیلم است. چنین جریانی ممکن است متشکل از یک جریان شکلی بهمراه دو جریان جهت انتقال صدای فیلم به صورت استریو و جریان چهارم هم حاوی زیرنویسی برای افراد ناشنوا یا ترجمه به زبانی غیر از زبان شکل باشد. در اینجا هم همگام سازی زیرجریانات اهمیت دارد. در صورت نادیده گرفتن همگام سازی، بازتولید فیلم غیرممکن خواهد بود. بعداً به این موضوع برگردیم.

<sup>599</sup> asynchronous transmission mode

<sup>600</sup> synchronous transmission mode

<sup>601</sup> end-to-end delay

<sup>602</sup> isochronous transmission mode

<sup>603</sup> simple stream

<sup>604</sup> complex stream

<sup>605</sup> substream

از نقطه نظر سیستم های توزیعی، پشتیبانی از جریانات مستلزم وجود المانهای مختلفی است . برای سهولت کار در اینجا روی پیوسته سازی داده های ذخیره شده در مقایسه با پیوسته سازی داده های زنده تمرکز می کنیم. در حالت اخیر، داده در زمان واقعی گرفته شده و از طریق شبکه به گیرنده ها ارسال می شود . تفاوت عمدی بین این دو آن است که پیوسته سازی داده های زنده فرمت کمتری برای تنظیم جریان در اختیار قرار می دهد. بنابراین با پیروی از Wu(۲۰۰۱) و گروه همکاران می توان طرح کلی معماري مشتری- سرور جهت پشتیبانی از جریانات پیوسته چندرسانه ای را مطابق شکل ۴-۲۶ ترسیم نمود.

معماری کلی ارائه شده چند مسأله مهم و قابل بحث را مطرح می کند. در درجه اول، داده های چندرسانه ای، خصوصاً شکل و تا حدودی صوت ، باید حدالامکان فشرده شوند تا بدین ترتیب ظرفیت ذخیره سازی لازم و خصوصاً ظرفیت شبکه کاهش یابد . نکته مهم تر از نقطه نظر ارتباطات ،کنترل کیفیت انتقال و مسائل همگام سازی است که در بخش بعد مورد بحث قرار گرفته است.

شكل ۴-۲۶-معماری کلی پیوسته سازی داده های چندرسانه ای ذخیره شده در یک شبکه (داخل شکل:داده های چندرسانه ای فشرده شده، سرور چندرسانه ای، کنترل QoS، مشتری، همگام سازی جریان، رمزگشایی جریان، رمزگشایی جریان، کنترل QoS ،شبکه).

#### ۴-۴-۲- جریانات و کیفیت سرویس

نیاز های زمانی (و دیگر نیازهای غیر کارکردي) غالباً به نام نیازهای کیفیت سرویس<sup>۶۰۶</sup> (QoS ) خوانده می شوند. این نیازها بیان می کنند که مثلًا برای تأمین روابط زمانی در یک رشته ، سیستم توزیعی و شبکه زیرین چگونه باید باشند. QoS، در جریانات داده ای پیوسته عمدتاً روی نیازهای متعهد به زمان بودن ، حجم و قابلیت اطمینان تأکید می کند. در این بخش، نگاه دقیقتري به QoS و نقش آن در ایجاد جریان خواهیم داشت.

راجع به نحوه بیان QoS موردنیاز مطالب بسیاری گفته شده است (بعنوان مثال مراجعه شود به Jin and Nahrstedt 2004، Jin and Nahrstedt ). این نیازها در برنامه کاربردی، در بسیاری از موارد تبدیل به چند خصوصیت مهم به ترتیب زیر میشود (Halsall ۲۰۰۱) :

۱. نرخ بیت لازم جهت انتقال داده .
۲. حد اکثر تأخیر تا زمان ایجاد جلسه (یعنی هنگامیکه یک برنامه کاربردی قادر به آغاز ارسال داده شود) .
۳. حد اکثر تاخیر انتها به انتها (یعنی چه مدت طول می کشد تا یک واحد داده بتواند به گیرنده برسد) .
۴. واریانس حد اکثر تأخیر، یا لرزش<sup>۶۰۷</sup> تأخیر .
۵. حد اکثر تأخیر سفر دوسویه<sup>۶۰۸</sup> .

لازم به ذکر است که همانطور که مثلًا در کتاب Steinmetz and Nahrstadt ( ۲۰۰۴ ) هم توضیح داده شده است، بهبودهای زیادی در این مشخصات قابل اعمال هستند . با این وجود، وقتی در مورد

<sup>606</sup> Quality of Service

<sup>607</sup> jitter

<sup>608</sup> round-trip

ارتباطات جریان گرا براساس محموله پروتکل اینترنت بحث می کنیم، باید این حقیقت را بپذیریم که ارتباط بر اساس سرویس بسیار ساده ای است، یعنی سرویس دیتاگرام بهترین تلاش: IP . هنگامی که در اجرا به مشکلی برミ خوریم، یعنی حالتی که در اینترنت بسیار به چشم می خورد، ویژگی IP به پیاده سازی پروتکل امکان می دهد تا هر زمان که لازم بداند بسته ها را حذف کند . تمام یا بسیاری از سیستم های توزیعی که از ارتباطات جریان گرا پشتیبانی می کنند، اکنون برروی محموله پروتکل اینترنت ساخته شده اند . بنابراین درمورد ویژگی های QoS ناگفته ها بسیار است (در واقع، هرچند IP تاحدوی از QoS پشتیبانی می کند، اما به ندرت پیاده سازی می شود) .

## اعمال QoS

با این فرض که سیستم زیربنایی فقط یک سرویس بهترین تلاش ارائه می دهد ، سیستم توزیعی می تواند عدم کیفیت سرویس را تا حد امکان مخفی نماید . خوشبختانه، بدین منظور از مکانیزم های مختلفی می توان استفاده نمود .

اولاً، وضعیت بهتر از چیزی است که تا اینجا ترسیم شده است. بعنوان مثال، اینترنت بوسیله سرویس های متمایز<sup>609</sup> خود، ابزاری جهت تمايز بین دسته بندی های مختلف داده ایجاد می کند. میزبان فرستنده اساساً قادر است تا بسته های خروجی را به عنوان متعلقات یکی از انواع دسته بندی ها علامتگذاری کند، بعنوان نمونه گروه ارسال پرشتاب<sup>610</sup> مشخص می کند که یک بسته باید بوسیله مسیریاب فعلی با در نظر گرفتن اولویت مطلق ارسال شود (David، ۲۰۰۲ و گروه همکاران) . بعلاوه، در دسته بندی ارسال تأیید شده<sup>611</sup>، ترافیک به چهار زیر دسته و سه روش برای حذف بسته ها در صورت مسدود بودن شبکه، تقسیم می شود . بنابراین ارسال تأیید شده به نحو مؤثری مجموعه ای از اولویتها قابل الصاق به بسته ها را تعریف کرده و به همین دلیل، به برنامه های کاربردی امکان می دهد تا بین بسته های حساس به زمان و بسته های غیرهم تمایز قائل شوند.

علاوه بر این راه حل های مربوط به شبکه، سیستم های توزیعی می توانند در رساندن داده ها به گیرنده ها هم مفید واقع شوند. هرچند معمولاً ابزارهای زیادی برای این منظور وجود ندارد، یک روش بسیار مفید استفاده از بافر جهت کاهش لرزش است. اصل حاکم بر این روش بسیار ساده بوده و در شکل ۴-۲۷ نمایش داده شده است. با این فرض که بسته ها در هنگام مخابره روی شبکه با واریانس خاصی تأخیر می کنند، توسط گیرنده تا حد اکثر زمان ممکنه در بافر ذخیره می شوند. این امر به گیرنده امکان می دهد تا با این فرض که همواره بسته های کافی به بافر وارد شده، بسته ها را تحت نرخ منظمی به برنامه کاربردی انتقال دهد و با همان نرخ پخش شود .

<sup>609</sup> differentiated services

<sup>610</sup> expedited forwarding

<sup>611</sup> assured forwarding

شکل ۴-۲۷- استفاده از بافر جهت کاهش لرزش (داخل شکل: بسته از منبع خارج می‌شود، بسته به بافر می‌رسد، بسته از بافر حذف می‌شود، زمان در بافر، شکاف در اجرایی مجدد، زمان بر حسب ثانیه).

البته ممکن است با اشکالاتی هم مواجه شویم، همانگونه که در بسته شماره ۸ در شکل ۴-۲۷ دیده می‌شود. ابعاد بافرگیرنده متناظر با ۹ ثانیه عبور بسته به برنامه کاربردی است. متأسفانه، رسیدن بسته شماره ۸ به گیرنده ۱۱ ثانیه طول کشید که در این فاصله زمانی بافر کاملاً تخلیه شده است. این امر باعث شکاف در اجرایی مجدد برنامه کاربردی می‌شود. تنها راه حل ممکنه افزایش ابعاد بافر می‌باشد. نقطه ضعف آشکار آن است که در این صورت، تأخیر آغاز پخش داده‌های موجود در بسته‌ها توسط برنامه کاربردی "دریافت" هم افزایش خواهد یافت.

روش‌های دیگری هم در این زمینه وجود دارد. با توجه به اینکه در مورد یک سرویس زیربنای بهترین تلاش بحث می‌کنیم، امکان گم شدن بسته‌ها وجود دارد. جهت جبران این نقصه در کیفیت سرویس، باید از تکنیکهای تصحیح خطأ استفاده کنیم (Wah، ۲۰۰۰ و گروه همکاران؛ ۱۹۹۸، Perkins و گروه همکاران). درخواست از فرستنده برای ارسال مجدد بسته مفقوده معمولاً امکان پذیر نیست، بنابراین باید از روش تصحیح خطای رو به جلو<sup>۶۱۲</sup> (FEC) استفاده شود. یک روش معروف، کدگذاری بسته خروجی به گونه‌ای است که دریافت  $k$  بسته از  $n$  بسته برای ساخت  $k$  بسته صحیح کفایت کند.

مشکل هنگامی ایجاد می‌شود که یک بسته حاوی چندین قالب صوتی و تصویری باشد. در نتیجه هنگامی که بسته ای مفقود می‌شود، عملآ دریافت کننده در هنگام اجرای قالبها به شکاف بزرگی برخورد می‌کند. این مشکل را می‌توان تا حدودی با قراردادن نوبتی قالبها، مطابق شکل ۴-۲۸ (ب)، از بین برد. به این ترتیب، در صورت مفقود شدن یک بسته، شکاف حاصله در میان قالبها پی در پی در زمان توزیع می‌شود. اما توجه داشته باشید که این روش در قیاس با روش‌های غیرنوبتی نیازمند بافر بزرگتری برای گیرنده است. بعنوان مثال، در شکل ۴-۲۸، گیرنده برای اجرای چهار قالب اول باید چهار بسته تحويل دهد، درحالیکه در روش ارسال غیر نوبتی این تعداد به یک کاهش می‌یابد.

شکل ۴-۲۸- تاثیر مفقود شدن بسته در (الف) ارسال غیرنوبتی و (ب) ارسال نوبتی (داخل شکل: (الف) بسته مفقود شده، ارسال شده، تحويل داده شده، شکاف ناشی از قالب‌های مفقود شده، بسته مفقودشده، (ب) ارسال شده، تحويل داده شده، قالب‌های مفقود شده).

#### ۴-۴-۳- همگام سازی جریان

یکی از مسائل مهم در ارتباط با سیستم‌های چندرسانه ای این است که جریانات مختلف که احتمالاً به شکل جریان پیچیده هستند، به صورت دوطرفه همگام می‌شوند. همگام سازی جریانات به معنی حفظ روابط زمانی بین جریانات است. دو نوع همگام سازی وجود دارد. ساده ترین شکل همگام سازی بین یک جریان داده ای گستره و یک جریان داده ای پیوسته است. برای نمونه، نمایش یک اسلاید را روی وب در نظر بگیرید که بوسیله صوت پشتیبانی شده است. هر اسلاید به شکل یک جریان داده ای گستره از سرور به مشتری انتقال

<sup>۶۱۲</sup> forward error correction

می یابد. در عین حال، مشتری باید (بخش مشخصی از) جریان صوتی را اجرا کند که با اسلاید حاضر هماهنگ بوده و از سرور هم پیش و اکنش شده باشد. در این حالت، جریان صوتی باید با ارائه اسلاید ها همگام شود.

نوع پرکاربردتر همگام سازی بین جریانات داده ای پیوسته انجام می شود. یکی از نمونه ها در زندگی روزمره نمایش فیلمی است که در آن جریان شکلی باید با صوت همگام شده و معمولاً به نام همگام سازی لبی<sup>۶۱۳</sup> خوانده می شود. نمونه دیگری از همگام سازی ها، اجرای یک جریان صوتی استریو متشکل از دو زیرجریان - یک جریان به ازای هر کانال- است. اجرای مناسب و موفقیت آمیز مستلزم آن است که هر دو زیرجریان کاملاً همگام باشند، بطوریکه وجود تفاوت بیش از  $2\text{ }\mu\text{sec}$  ممکن است باعث ازدست رفتن تأثیر استریو شود. همگام سازی در سطح واحدهای داده ای تشکیل دهنده جریان رخ می دهد. به بیان دیگر، دو جریان فقط در بین واحدهای داده ای قابل همگام سازی هستند. درک واقعی یک واحد داده ای تا حد زیادی بستگی به سطح انتزاع انتخابی برای مشاهده جریان داده دارد. برای درک بهتر این موضوع، مجدداً مثال جریان صوتی (تک کانالی) با کیفیت CD را بیاد بیاورید. دربهترین حالت، این نوع جریان به صورت رشته ای از نمونه های ۱۶ بیتی به نظر می رسد. تحت فرکانس نمونه گیری  $44100\text{ Hz}$ ، همگام سازی با دیگر جریانات صوتی می تواند، در تئوری، تقریباً هر  $23\text{ }\mu\text{sec}$  رخ دهد. برای ارتقاء کیفیت استریو، همگام سازی در این حد واقعاً ضروری است.

با این وجود، هنگامی که همگام سازی بین یک جریان صوتی و یک جریان تصویری را برای همگام سازی لبی در نظر می گیریم، همگام سازی در گام های بلندتری انجام می شود. همانطور که قبلاً هم گفتیم، فریم های ویدئویی باید تحت سرعت  $25\text{ Hz}$  یا بیشتر اجرا شوند. با انتخاب استاندارد پرکاربرد  $29/97\text{ Hz}$ ،  $NTSC\text{ ۲۹/۹۷}\text{ Hz}$ ، می توانیم نمونه های صوتی را در واحدهای منطقی دسته بندی کنیم که در طی زمان اجرای فریم ویدئو ( $33\text{ msec}$ ) ادامه پیدا کند. بنابراین، تحت فرکانس  $44100\text{ Hz}$  نمونه گیری صوت، ابعاد یک واحد داده صوتی می تواند) با این فرض که هر نمونه ۱۶ بیت است)، به اندازه  $1470\text{ نمونه}$  یا  $11760\text{ بایت}$  باشد. در عمل، واحدهای بزرگتر تا  $40$  یا حتی  $80\text{ msec}$  هم امکان پذیر است(Steinmetz).

### مکانیزم های همگام سازی

اجازه دهید در مورد نحوه انجام همگام سازی صحبت کنیم. لازم است دو موضوع را از هم تفکیک کنیم: **(الف)** مکانیزم های اصلی جهت همگام سازی دو جریان و **(ب)** توزیع مکانیزم ها در محیط شبکه.

مکانیزم های همگام سازی را می توان در لایه های انتزاع متعدد و مختلف تصور نمود. در پایین ترین سطح، همگام سازی به صورت آشکار با عملیات روی واحدهای داده ای جریانات ساده انجام می پذیرد. این مطلب در شکل ۴-۲۹ نمایش داده شده است. به طور خلاصه، فرآیندی وجود دارد که به سادگی عملیات خواندن و نوشتن

<sup>613</sup> lip synchronization

را روی جریانات ساده مختلف انجام داده و تضمین می کند که این عملیات با محدودیتهاي همگام سازی و زمانبندی خاص مطابقت دارد.  
شکل ۴-۲۹-اصل کلی همگام سازی آشکار روی واحدهای داده سطح (داخل شکل: ماشین گیرنده، برنامه کاربردی، روال خواندن دو واحد داده صوتی به ازای هر واحد داده شکلی، جریان ورودی، شبکه).

بعنوان مثال فیلمی را در نظر بگیرید که به صورت دو جریان ورودی ارائه شده است. جریان شکل حاوی تصاویر کم کیفیت متراکم نشده  $320 \times 240$  پیکسل با یک بایت کد شده و در نتیجه واحدهای داده تصویری  $76800$  بایت را بوجود آورده است. فرض کنید که قرار است تصاویر تحت Hz  $30$  با به صورت یک شکل در هر  $32$  msec نمایش داده شود. فرض شده است که جریان صوتی حاوی نمونه های صوتی است که در گروه های واحدهای  $11760$  بایتی قرار گرفته و مطابق شرح فوق، هر یک متناظر با  $32$  msec صوت می باشد. چنانچه روال ورودی قادر به هدایت  $2/7$  MB/sec باشد، همگام سازی لبی به راحتی با اجرای نوبتی خواندن شکل و خواندن نمونه های بلوک صوتی هر  $32$  ثانیه یکبار امکان پذیر است.

نقشه ضعف روش آن است که برنامه کاربردی به طور کامل مسئول اجرای همگام سازی می باشد و در عین حال از امکانات سطح پایینی برای اینکار برخوردار است. روش بهتر ارائه رابطی برای برنامه کاربردی است که امکان کنترل بهتر جریانات و وسیله ها را فراهم آورد. به مثال قبل باز می گردیم. فرض کنید که نمایش شکل رابط کنترلی دارد که بوسیله آن می توان نرخ نمایش شکل را تعیین نمود. بعلاوه، رابط امکاناتی برای ثبت یک عمل کننده قابل تعریف بوسیله کاربر ارائه می دهد که به محض ورود  $k$  شکل جدید، فرآخوانی می شود. وسیله صوتی هم رابطی مشابه همین ارائه می دهد. با داشتن این رابط های کنترلی، نویسنده برنامه کاربردی می تواند یک برنامه ساده نظارتی متشکل از دو عمل کننده - یعنی یکی به ازای هر جریان- بنویسد؛ ایندو با همکاری هم چک می کند که آیا جریان صوت و شکل به اندازه کافی همگام شده اند یا نه و در صورت نیاز، سرعت اجرای واحدهای صوت و شکل را هماهنگ می کند.

مثال اخیر در شکل ۴-۳۰ ارائه شده و نمونه ای از سیستم های میان افزاری چندرسانه ای است. به طور خلاصه، میان افزار چندرسانه ای مجموعه ای از رابط ها را برای کنترل جریانات صوتی و شکلی از قبیل رابط های مربوط به کنترل وسیله ها از قبیل مانیتورها، دوربین ها، میکروفون ها و غیره ارائه می دهد. هر وسیله و جریان رابط های سطح بالای خاص خود از قبیل رابط های مربوط به مطلع کردن برنامه کاربردی در هنگام وقوع رخدادی خاص را دارد. از این رابط ها بعداً جهت نوشتن عمل کننده ها برای همگام سازی جریان ها استفاده شده است. نمونه هایی از این دست رابط ها در کتاب (Stefani ۱۹۹۸) آمده است.

شکل ۴-۳۰- اصل کلی همگام سازی که در رابط های سطح بالا پشتیبانی می شود (داخل شکل: ماشین گیرنده، برنامه کاربردی، برنامه کاربردی به میان افزار می گوید که با جریانات ورودی چگونه برخورد کند، کنترل چندرسانه ای بخشی از میان افزار است، لایه میان افزار، جریان ورودی، شبکه).

توزيع مکانیزم های میان افزاری مسئله قابل توجه دیگری است. اولاً، طرف دریافت کننده یک جریان پیشرفته که از چندین زیرجریان نیازمند همگام سازی تشکیل شده ، باید دقیقاً از وظیفه خود اطلاع داشته باشد. به بیان دیگر، توصیف همگام سازی کامل آن به صورت محلی موجود باشد. روش معمول این است که اطلاعات به صورت ضمنی با مالتی پلکسی کردن<sup>۶۱۴</sup> جریانات مختلف به یک جریان واحد که شامل تمامی واحدهای داده ای از جمله واحدهای داده ای مربوط به همگام سازی باشد، انجام پذیرد.

در جریانات MPEG از روش اخیر جهت همگام سازی استفاده می شود . استانداردهای MPEG (گروه متخصصین شکل متحرک)<sup>۶۱۵</sup> مجموعه ای از الگوریتم ها را برای فشرده سازی صوت و شکل ارائه می دهد. استانداردهای MPEG متنوع هستند. بعنوان مثال، MPEG-2 اساساً برای فشرده سازی تصویر با کیفیت پخش از ۶ Mbps تا ۴ طراحی شده است. در این شیوه ، تعداد نامحدودی از جریانات پیوسته و گستره را می توان در یک جریان واحد قرار داد . هر جریان ورودی، ابتدا به جریانی از بسته ها تبدیل می شود که براساس ساعت سیستم kHz ۹۰ ، مهر زمانی قابل حمل است . سپس این جریانات به جریان برنامه ای<sup>۶۱۶</sup> مالتی پلکس می شوند که متشکل از بسته هایی با طول متغیر است ، ولی همگی دارای مبنای زمانی واحدی هستند. طرف گیرنده با استفاده از مهرهای زمان هر بسته بعنوان مکانیزم پایه برای همگام سازی بین جریانی، اقدام به جداسازی جریان ها می کند.

مسئله مهم دیگر این است که آیا همگام سازی باید در طرف گیرنده انجام شود یا در طرف فرستنده . چنانچه همگام سازی توسط فرستنده صورت پذیرد، جریان های دارای انواع مختلف واحدهای داده ای را می توان در یک جریان واحد قرار داد . مجدداً جریان صوتی استریو را در نظر بگیرید که به ازای هر کanal، از دو زیر جریان تشکیل شده است. یک روش، انتقال مستقل هر جریان به گیرنده است و اینکه اجازه دهیم گیرنده، نمونه ها را به صورت جفت آگاه همگام کند. مشخص است که چون ممکن است هر زیرجریان دچار تأخیرهای مختلفی شود، احتمالاً همگام سازی به سختی انجام خواهد شد. روش بهتر، قرار دادن هردو زیرجریان در یک جریان در طرف فرستنده است. جریان حاصله شامل واحدهای داده ای متشکل از جفت نمونه هاست که به ازای هر کanal یک نمونه وجود دارد . اکنون گیرنده باید صرفاً واحد داده ای مورد نظر را خوانده و سپس آنرا بین جریان چپ و راست تقسیم کند. در اینصورت تأخیر در هر دو کanal برابر خواهد بود.

#### ۶-۴-۵- ارتباط چند پخشی

یکی از موضوعات مهم در ارتباطات سیستم های توزیعی ، پشتیبانی از ارسال داده به چند گیرنده یا همان ارتباط چند پخشی است. سالهای است که این موضوع در حوزه پروتکل های شبکه قرار گرفته و پیشنهادات متعددی برای راه حل های سطح انتقال و سطح شبکه

<sup>614</sup> multiplexing

<sup>615</sup> Motion Picture Experts Group

<sup>616</sup> program stream

<sup>617</sup> multicast communication

پیاده سازی و ارزیابی شده اند (Janic، ۱۹۹۸؛ Obraczka، ۲۰۰۵). یک مسئله مهم در تمامی راه حل ها، ایجاد مسیرهای ارتباطی جهت انتشار اطلاعات بوده است. در عمل، تحقیق این ایده مستلزم تلاش های مدیریتی فراوان و در بسیاری موارد، دخالت های انسانی بود. بعلاوه، مدامی که تمامی پیشنهادات روی یک موضوع واحد همگرایی نداشته باشند، ISP‌ها تمایل چندانی به پشتیبانی از چندپخشی خواهند داشت (Diot، ۲۰۰۰).

همگام با ظهور فن آوری نظیر به نظر و بخصوص، مدیریت فوقانی ساخت یافته، راه برای ایجاد مسیرهای ارتباطی هموارتر شد. از آنجایی که راه حل های نظیر به نظر نوعاً در لایه کاربردی اجرا می شوند، روش‌های چندپخشی سطح برنامه کاربردی متعددی معرفی شده اند. در این بخش، نظری به این تکنیک‌ها خواهیم داشت.

ارتباط چندپخشی به روش‌های دیگری غیر از ایجاد مسیرهای ارتباطی آشکار هم قابل انجام است. همانطور که در این بخش هم خواهیم دید، انتشار اطلاعات براساس شایعه<sup>۶۱۸</sup> روش‌های ساده ای (هرچند غالباً با اثربخشی کمتر) را جهت چندپخشی ایجاد می کند.

#### ۱-۵-۴-چندپخشی در سطح کاربردی

ایده اصلی در چندپخشی سطح کاربردی این است که گره‌ها به شکل یک شبکه فوقانی سازماندهی شده و سپس جهت انتشار اطلاعات به اعضای خود بکار برده شوند. یکی از مشاهدات مهم این است که مسیریاب‌های شبکه در عضویت‌های گروهی سهیم نمی شوند. در نتیجه، اتصالات بین گره‌ها در شبکه فوقانی ممکن است از پیوندهای فیزیکی متعددی عبور کند و به همین دلیل، پیام‌های مسیریابی در داخل شبکه فوقانی ممکن است، در قیاس با آنچه بوسیله مسیردهی سطح شبکه بوجود می آید بهینه نباشد.

یکی از مسائل بسیار مهم در طراحی، ساخت شبکه فوقانی است. به طور خلاصه، دو روش در این رابطه وجود دارد (El-Sayed، ۲۰۰۳). در حالت اول، ممکن است گره‌ها مستقیماً به شکل یک درخت سازماندهی شود که در اینصورت فقط یک مسیر(فوقانی) در بین هر جفت گره وجود خواهد داشت. در حالت دوم ممکن است گره‌ها به صورت یک شبکه مشبك<sup>۶۱۹</sup> سازماندهی شوند که در آن، هر گره دارای چندین همسایه بوده و در کل بین هر جفت گره چندین مسیر وجود خواهد داشت. تفاوت عمده بین این دو روش آن است که حالت اخیر مقاوم تر است؛ یعنی چنانچه یکی از اتصالات بشکند (مثلاً در اثر خرابی گره)، باز هم این امکان وجود دارد که بدون نیاز به سازماندهی مجدد و فوری کل شبکه فوقانی، اقدام به انتشار اطلاعات نمود.

برای درک بهتر موضوع، اجازه دهید طرح نسبتاً ساده تر ساخت درخت چندپخشی در Chord را در نظر بگیریم که در فصل ۲ شرح داده شد. این طرح که برای اولین بار در Castro، Scribe<sup>۲۰۰۲</sup> و گروه (Rowstorn and Druschel، ۲۰۰۱) پیشنهاد شد، طرح چندپخشی سطح برنامه کاربردی است که در بالای Pastry (Pastry و Druschel) ساخته شده است. یک سیستم نظیر به نظر بر اساس DHT می باشد.

<sup>618</sup> gossip-based

<sup>619</sup> mesh network

تصور کنید که گره مفروض قصد آغاز جلسه چندپخشی را داشته باشد. بدین منظور، یک شناسه چندپخشی ، مثلاً  $mid$  را ایجاد می کند که صرفاً یک کلید ۱۶۰ بیتی تصادفاً انتخاب شده است. سپس  $succ(mid)$  را جستجو کرده که در واقع گره مسئول کلید موردنظر است و آنرا ارتقاء می دهد تا تبدیل به ریشه درخت چندپخشی شود که جهت ارسال داده ها به گره های مورد علاقه استفاده خواهد شد. گره  $P$  جهت  $LOOKUP(mid)$  می پیوستن به درخت، فقط اقدام به اجرای عملیات  $(mid)$  نماید. به این ترتیب، یک پیغام جستجو با درخواست برای پیوستن به گروه چندپخشی  $mid$  از  $P$  به  $succ(mid)$  مسیرده خواهد شد. همانطور که قبلاً هم گفته شد، راجع به خود الگوریتم مسیرده بتفصیل در فصل ۵ صحبت خواهیم کرد.

درخواست پیوستن در طی مسیر خود به سمت ریشه، از گره های متعددی عبور خواهد کرد. فرض کنید که ابتدا به گره  $O$  برسد. چنانچه  $Q$  تا پیش از این لحظه با درخواست پیوستن به  $mid$  روبرو نشده باشد، تبدیل به **پیش برنده<sup>۶۲۰</sup>** برای آن گروه خواهد شد. در اینجا،  $P$  فرزند  $Q$  خواهد شد و  $Q$  به پیش بردن درخواست پیوستن به ریشه ادامه خواهد داد. چنانچه گره بعدی ریشه، مثلاً  $R$  هم پیش برنده گروه نباشد، تبدیل به پیش برند شده و پس از ثبت  $Q$  بعنوان فرزند خود ، به پیش بردن درخواست پیوستن ادامه خواهد داد.

از طرف دیگر، چنانچه  $Q$  (یا  $R$ ) از قبل پیش برنده  $mid$  باشد، آنهم پیش برنده قبلی (یعنی به ترتیب،  $P$  یا  $Q$ ) را بعنوان فرزند خود ثبت می کند، اما دیگر نیازی به ارسال درخواست پیوستن به ریشه وجود نخواهد داشت؛ چون  $Q$  (یا  $R$ ) از قبل عضو درخت چندپخشی هستند.

گره هایی از قبیل  $P$  که آشکارا درخواست پیوستن به درخت چندپخشی را نموده اند، بر اساس تعریف پیش برنده نیز خوانده می شوند. نتیجه طرح این است که ما یک درخت چندپخشی در شبکه فوقانی با دو نوع گره ایجاد می کنیم : گره های صرفاً پیش برنده که به عنوان کمک رسان<sup>۶۲۱</sup> عمل می کنند، و گره هایی که آنها هم پیش برنده هستند اما صراحتاً خواستار پیوستن به درخت هم می باشند . حال چندپخشی ساده می شود: یک گره با اجرای مجدد عملیات  $LOOKUP(mid)$  ، صرفاً یک پیام چندپخشی را به ریشه درخت ارسال کرده و سپس می توان این پیام را به تمام درخت گسیل نمود. اشاره می کنیم که تشریح سطح بالای چندپخشی در Scribe شباهت چندانی به طرح اصلی آن ندارد. بنابراین از خوانندگان مشتاق درخواست می شود برای مطالعه تفصیلی تر به کتاب (Castro ۲۰۰۲) و گروه همکاران مراجعه کنند.

### ساخت شبکه فوقانی

با توجه به شرح سطح بالای فوق الذکر، به راحتی می توان دریافت که هرچند ساخت درخت به تنها یک در حالی که گره ها به شبکه فوقانی سازماندهی شده اند مشکل نیست، ولی ساخت درخت مفید و مؤثر مسأله ای جدا و متفاوت است. توجه داشته باشید که تا به

<sup>620</sup> forwarder

<sup>621</sup> helper

اینجا، برای انتخاب گره هایی که در ساخت درخت مشارکت می کنند، هیچ معیار کارآیی لحاظ نشده است. به این معنا که اینکار فقط بر اساس مسیردهی (منطقی) پیام ها از طریق شبکه فوقانی انجام شده است.

شكل ۴-۳۱-روابط بین پیوندها در یک شبکه فوقانی و مسیرهای سطح شبکه ای واقعی (داخل شکل: میزبان نهایی، مسیریاب، شبکه فوقانی، اینترنت).

برای درک مشکل مطرح شده، نگاهی به شکل ۴-۳۱ می اندازیم که در آن مجموعه کوچکی از چهار گره به شکل یک شبکه فوقانی ساده سازماندهی شده و گره A ریشه درخت چندپیخشی را تشکیل می دهد. هزینه های حرکت در یک پیوند فیزیکی هم ارائه شده است. مشاهده می کنید که هر زمانیکه A پیامی را به گره های دیگر چندپیخشی می کند، این پیام دوبار از هریک از پیوندهای  $B, Rb$ ,  $Ra, Rb$ ,  $D, Rd$ ,  $Rc, Rd$ ,  $B, Rb$ ,  $Ra, Rb$  عبور خواهد کرد. در صورتی که به جای ایجاد پیوند شبکه فوقانی از B به D، پیوند را از A به C ایجاد می کردیم، کارآیی شبکه فوقانی افزایش می یافتد. تحت چنین پیکربندی دیگر نیازی به پیمودن دو مرتبه ای پیوندهای  $Ra, Rb$ ,  $Rc, Rd$ ,  $Rd, Ra$  و  $Rb, Rb$  وجود نخواهد داشت.

کیفیت درخت چندپیخشی سطح برنامه کاربردی غالباً بر اساس سه معیار مختلف یعنی فشار پیوند<sup>۶۲۲</sup>، کشیدگی<sup>۶۲۳</sup> و هزینه درخت<sup>۶۲۴</sup> برآورد می شود. فشار پیوند به ازای هر پیوند تعریف شده و شمارش می کند که یک بسته، چندبار از پیوند مورد نظر عبور خواهد کرد (Chu, ۲۰۰۲)، گروه همکاران). فشار پیوند بیشتر از یک به این معناست که هر چند یک بسته ممکن است در سطح منطقی در امتداد دو اتصال مختلف ارسال شود، بخشی از این اتصالات ممکن است واقعاً متناظر با یک پیوند فیزیکی واحد بوده که نمونه ای از آنرا در شکل ۴-۳۱ مشاهده کردیم.

**کشیدگی یا جریمه تأخیر نسبی (RDP)** حاصل تقسیم تأخیر بین دو گره در شبکه فوقانی بر تأخیری است که آندو گره در شبکه زیرین تجربه خواهد کرد. بعنوان مثال، در شبکه فوقانی مفروض، پیام ها برای عبور از B به C مسیر  $B \rightarrow Rb \rightarrow Ra \rightarrow Rc \rightarrow C$  را طی می کند که در مجموع ۵۹ واحد هزینه دارد. با این وجود، پیام ها در شبکه زیرین در امتداد مسیر  $B \rightarrow Rb \rightarrow Rd \rightarrow Rc \rightarrow C$  با هزینه کلی ۴۷ واحد مسیردهی می شوند که در نهایت باعث ایجاد کشیدگی<sup>۶۲۵</sup> ۱/۲۲۵ می شود. مشخص است که در ساخت شبکه های فوقانی، هدف به حداقل رساندن کشیدگی کلی یا به بیان دیگر، میانگین RDP برآورد شده روی تمام جفت گره هاست.

در نهایت، هزینه درخت یک مقیاس جهانی بوده و غالباً با به حداقل رساندن جمع هزینه های پیوند ارتباط دارد. بعنوان نمونه، در صورتی که هزینه یک پیوند، تأخیر بین دو گره نهایی آن فرض شود، آنگاه بهینه سازی هزینه درخت تبدیل به پیدا کردن درخت

<sup>622</sup> link stress

<sup>623</sup> stretch

<sup>624</sup> tree cost

<sup>625</sup> Relative Delay Penalty

فاصله زمانی مینیمم<sup>۶۲۶</sup> خواهد شد که در آن، مجموع زمان انتشار اطلاعات به همه گره ها حداقل است.

برای ساده سازی مطالب ، فرض کنید که یک گروه چندپیشی دارای گره شناخته شده ای است که گزارش گره هایی را نگهداری می کند که به درخت پیوسته اند. هنگامی که یک گره جدید درخواست پیوستن می دهد ، با این گره قرار ملاقات<sup>۶۲۷</sup> تماس برقرار می کند تا لیست (احتمالاً تقریبی) اعضا را بدست آورد. هدف از اینکار انتخاب بهترین عضوی است که بتواند بعنوان ولی گره جدید در درخت عمل کند. اما چه کسی انتخاب خواهد شد؟ راههای فراوان و پیشنهادات مختلفی وجود دارند که غالباً راه حل های بسیار متفاوتی را پی می گیرند.

بعنوان مثال، یک گروه چندپیشی با یک منبع واحد را درنظر بگیرید. در این حالت، انتخاب بهترین گره به راحتی انجام خواهد شد: در واقع همان منبع است (چون در این حالت می توانیم مطمئن باشیم که کشیدگی برابر با ۱ خواهد بود). به این ترتیب ، یک توبولوژی ستاره ای مطرح کرده ایم که منبع در میان آن قرار گرفته است. اگر چه ساختار ساده ای است ، ولی به راحتی می توان تصور کرد که منبع دچار اضافه بار خواهد شد. به بیان دیگر، انتخاب گره غالباً به صورتی انجام خواهد شد که فقط گره هایی انتخاب شوند که  $k$  همسایه یا کمتر داشته باشند-  $k$  در واقع پارامتر طراحی است. این محدودیت به شدت باعث پیچیدگی الگوریتم ایجاد درخت می شود؛ چون ممکن است یکی از راه حل های مناسب پیشنهادی شامل پیکربندی مجدد درخت فعلی باشد.

Tan (۲۰۰۳) و گروه همکاران در کتاب خود مرور و ارزیابی جامعی بر راه حل های مختلف این مشکل ارائه داده اند. برای درک بهتر، بیایید نگاه دقیق تری به یک خانواده خاص به نام درختهای تغییر<sup>۶۲۸</sup> (Helder and Jamin, ۲۰۰۲)، بیندازیم . ایده اساسی بسیار ساده است. فرض کنید که اکنون یک درخت چندپیشی با یک منبع واحد بعنوان ریشه داریم. در این درخت، گره  $P$  می تواند با قطع پیوند ولی فعلی خود و ایجاد اتصال با گره دیگر، ولی را تغییر دهد. تنها محدودیت در تغییر پیوندها آن است که ولی جدید دیگر هرگز نمی تواند عضو زیردرخت با ریشه  $P$  باشد (چون اینکار باعث بخش بندی درخت و ایجاد حلقه خواهد شد) و بعلاوه ولی جدید دیگر دارای فرزندان بلافصل زیادی نخواهد بود. مورد اخیر لازم است تا بار پیام های ارسالی توسط هریک از گره ها محدود شود.

تصمیم گیری راجع به تغییر ولی تابع ضوابط مختلفی است. یکی از ساده ترین آنها بهینه سازی مسیر به سمت منبع است که به نحو مؤثری، باعث به حداقل رساندن تأخیر در هنگامی می شود که قرار است پیامی چندپیشی شود. بدین منظور، هر گره به صورت منظم اطلاعات روی گره های دیگر را دریافت می کند (یکی از روشهای خاص اینکار را ذیلاً شرح خواهیم داد). در اینجا، گره می تواند بررسی کند که آیا گره دیگر از نظر تأخیر در طول مسیر به سمت منبع

<sup>626</sup> minimal spanning tree

<sup>627</sup> rendezvous node

<sup>628</sup> switch trees

ولی بهتری خواهد بود و در صورت مثبت بودن، کار تغییر را آغاز خواهد کرد.

معیار دیگر این است که آیا تأخیر به ولی احتمالی دیگر کمتر از ولی فعلی خواهد بود . اگر هر یک از گره ها این ضابطه را رعایت کنند، آنگاه کل تأخیر درخت حاصله در حالت ایده آل باقیستی حداقل باشد. به بیان دیگر، اینکار نمونه ای از بهینه سازی هزینه درخت است که در بالا شرح داده شد. با این وجود، برای ساخت چنین درختی نیازمند اطلاعات بیشتری هستیم. اما همچنانکه مشاهده خواهیم کرد، همین طرح ساده ابتکاری منطقی است که تخمین خوبی از یک درخت حداقل فاصله زمانی ارائه می دهد.

بعنوان نمونه، حالتی را درنظر بگیرید که در آن گره  $P$  اطلاعات همسایگان ولی خود را دریافت می کند. توجه داشته باشید که در این حالت همسایه ها شامل جد  $P$  به همراه خواهر و برادرهای ولی  $P$  می باشد . بنابراین گره  $P$  می تواند تأخیرهای هر یک از این گره ها را ارزیابی کرده و در نهایت، گره دارای کمترین تأخیر مثلاً  $Q$  را بعنوان ولی جدید خود انتخاب کند. بدین منظور، درخواست تغییر را برای  $Q$  ارسال می کند. جهت جلوگیری از ایجاد حلقه در اثر همزمانی درخواست های تغییر، گره دارای درخواست تغییر معوق پردازش تمامی درخواستهای واردہ را رد خواهد کرد. در واقع، اینکار باعث می شود که فقط تغییرات کاملاً مستقل را بتوان بصورت همزمان اجرا کرد. بعلاوه،  $P$  اطلاعات کافی در اختیار  $Q$  خواهد گذاشت تا  $Q$  بتواند نتیجه بگیرد که هردو گره ولی واحدی داشته یا اینکه ،  $Q$  جد است.

یکی از مشکلات مهمی که تا به اینجا بحثی از آن به میان نیامد، خرابی گره<sup>۶۲۹</sup> است. در مورد درخت های تغییر، یک راه حل ساده پیشنهاد می شود: هر وقت که گرهی از خرابی ولی خود اطلاع پیدا کند، خود را به ریشه متصل می کند. در این حالت، پروتکل بهینه سازی مطابق معمول پیش خواهد رفت و در نهایت، گره را در نقطه مناسبی در درخت چندپیشی قرار خواهد داد. تجربیات ذکرشده در کتاب (۲۰۰۲) Helder and Jamin نزدیک به درخت فاصله زمانی مینیمم است.

#### ۴-۵-۲- انتشار داده ها بر اساس شایعه

یکی از روشهای فوق العاده مهم در انتشار اطلاعات براساس رفتار /پیدمیک<sup>۶۳۰</sup> انجام می شود. بر اساس نحوه شیوع بیماری ها در بین انسان ها، محققان از مدت‌ها پیش تحقیق برروی استفاده از تکنیک های ساده جهت توسعه و گسترش اطلاعات در سیستم های توزیعی پردازنه را آغاز نموده اند. هدف اصلی این پروتکل های اپیدمیک<sup>۶۳۱</sup> پخش سریع اطلاعات در بین مجموعه های بزرگی از گره ها با استفاده از اطلاعات صرفاً محلی است. به بیان دیگر، برای هماهنگ سازی انتشار اطلاعات، هیچ جزء مرکزی وجود ندارد. جهت تشریح اصول کلی این دست الگوریتم ها، فرض می کنیم که تمامی به روزآوری ها برای یک آیتم داده ای فقط در یک گره آغاز می شود. به این ترتیب، به راحتی می توان از تداخل نوشتن- نوشتن

<sup>629</sup> node failure

<sup>630</sup> epidemic behaviour

<sup>631</sup> epidemic protocols

اجتناب کرد . مطلب زیر براساس یک مقاله کلاسیک اثر Demers و گروه همکاران در سال ۱۹۸۷ درمورد الگوریتم های اپیدمیک ارائه شده است. یکی از بررسی های اخیر در مورد انتشار اطلاعات اپیدمیک را می توانید در کتاب (۲۰۰۴) Eugster و گروه همکاران مطالعه کنید.

## مدل های انتشار اطلاعات

همانطور که از نام آن بر می آید، الگوریتم های اپیدمیک براساس تئوری اپیدمی ها استوار است که به مطالعه شیوع بیماری های عفونی می پردازد. در مورد سیستم های توزیعی پرداخته، این الگوریتم ها به جای پراکندن بیماری ها، باعث پراکندن اطلاعات می شوند. تحقیق در مورد اپیدمی ها در سیستم های توزیعی با هدف کاملاً متفاوتی دنبال شده است. هرچند سازمان های بهداشتی نهایت تلاش خود را برای جلوگیری از شیوع بیماریهای عفونی در بین گروه های بزرگ انسانی انجام می دهند، طراحان الگوریتم های اپیدمیک در سیستم های توزیعی تلاش می کنند تا تمامی گره های دارای اطلاعات جدید را با سرعت هرچه تمامتر "عفونی" کنند.

با استفاده از اصطلاح های مورد استفاده در بیماری های اپیدمیک، گرهي که بخشی از یک سیستم توزیعی محسوب می شود، در صورت داشتن اطلاعاتی که بخواهد به گره های دیگر انتشار یابد، **عفونی**<sup>۶۳۲</sup> و گرهي که تا کنون این اطلاعات را ندیده است، **در معرف**<sup>۶۳۳</sup> نامیده می شود. در نهایت ، گره به روزآوری شده ای که قصد یا توان توزیع داده های خود را ندارد، **حذف شده**<sup>۶۳۴</sup> نامیده می شود. البته در اینجا فرض بر این است که می توان مابین داده های قدیم و جدید تفاوت قائل شد ، مثلاً به این دلیل که داده ها مهر زمانی خورده یا نسخه بندی شده اند . از اینرو گفته می شود که گره ها به روزآوری ها را هم توزیع می کنند.

یکی از مدل های پخش معروف ، مدل **ضدآنتروپی**<sup>۶۳۵</sup> است. در این مدل، یک گره  $P$  گره دیگر  $Q$  را به صورت تصادفي انتخاب کرده و سپس به روزآوری ها را با  $Q$  مبادله می کند. سه روش برای تبادل به روزآوری ها وجود دارد:

۱.  $P$  صرفاً به روزآوری های خود را در  $Q$  هل می دهد<sup>۶۳۶</sup> .

۲. فقط به روزآوری های خود را از  $Q$  می کشد<sup>۶۳۷</sup> .

۳.  $P$  و  $Q$  به روزآوری های خود را برای هم می فرستند ( یعنی روش هل دادن- کشیدن).

در زمان انتشار سریع به روزآوری ها ، تنها روش نامناسب از بین سه روش فوق ، به روزآوری های هل دادن است . برای درک موضوع به این موارد توجه کنید. اولاً) در یک روش صرفاً براساس هل دادن، به روزآوری ها فقط از طریق گره های عفونی قابل پخش است . با این وجود، چنانچه تعداد گره های زیادی عفونی شوند، احتمال اینکه هریک گره در معرضی را انتخاب نماید، نسبتاً ناچیز

<sup>632</sup> infected

<sup>633</sup> susceptible

<sup>634</sup> removed

<sup>635</sup> anti-entropy

<sup>636</sup> push

<sup>637</sup> pull

است. در نتیجه، این احتمال وجود دارد که یک گره خاص فقط به این دلیل که توسط گره عفونی انتخاب نشده، برای مدت زمان طولانی در معرض باقی بماند.

بالعکس، روش براساس کشیدن در صورتی بهترین کارآیی را خواهد داشت که تعداد گره های زیادی عفونی شده باشند. در این حالت، پخش به روزآوری ها اساساً بواسیله گره های در معرض انجام می شود. احتمال زیادی وجود دارد که چنین گرهی با یک گره عفونی تماس برقرار کرده و بعداً به روزآوری ها را درون خود کشیده و آن را هم عفونی کند.

می توان نشان داد که در صورت عفونی شدن فقط یک گره، به روزآوری ها با استفاده از هر یک از اشکال ضدآنترپوپی به سرعت در تمامی گره ها انتشار خواهند یافت، هرچند روش کشیدن- هل دادن بازهم بهترین روش خواهد بود (Jelarsity<sup>638</sup> و گروه همکاران). یک دور<sup>639</sup> به صورت یک فاصله زمانی تعریف می شود که در آن هر گره حداقل یک مرتبه برای اولین بار اقدام به تبادل به روزآوری ها با یک گره تصادفاً انتخاب شده دیگر می نماید. بنابراین می توان نشان داد که تعداد دورها برای انتشار به روزآوری مفروض در تمام گره ها ( $O(\log N)$ ) طول می کشد که در آن  $N$  تعدادگره ها در سیستم است. در واقع بیان می کند که انتشار به روزآوری ها به سرعت و مهم تر از آن، بصورت مقیاس پذیر انجام می شود.

یکی از نمونه های خاص این روش انتشار شایعه<sup>640</sup> یا به بیان ساده تر شایعه پراکنی<sup>641</sup> است. در این روش اگر گره  $P$  به تازگی برای آیتم داده ای  $x$  به روزآوری شده باشد، با گره اختیاری دیگر یعنی  $Q$  تماس برقرار کرده و تلاش می کند تا به روزآوری را به  $Q$  هل دهد. اما این امکان وجود دارد که  $Q$  پیش از آن به وسیله گره دیگری به روزآوری شده باشد. در این حالت،  $P$  ممکن است مثلاً با احتمال  $1/k$  تمایل خود را به پخش بیشتر از دست دهد. به بیان دیگر، از آن به بعد حذف خواهد شد.

شایعه پراکنی از بسیاری جهات مشابه زندگی روزمره ماست. وقتی باب اخبار داغی برای مطرح کردن با دیگران داشته باشد، ممکن است با زنگ زدن به دوست خود آليس همه چیز را به او بگوید. آليس هم مثل باب ممکن است دوستان نزدیکی داشته باشد که بخواهد اخبار را به آنها بگوید. با این وجود، ممکن است بعد از زنگ زدن به یکی از دوستان خود به نام چاک و اطلاع از اینکه او قبل اخبار را شنیده است، از زنگ زدن به بقیه دوستان خود و خبررسانی به آنها منصرف شود. چون شنیدن اخبار دست دوم هیچ لذتی برای کسی ندارد.

بنظر می رسد شایعه پراکنی یکی از بهترین راه ها برای پخش سریع اخبار باشد. با این وجود، نمی توان تضمین کرد که تمامی گره ها واقعاً به روزآوری شوند (Demers, ۱۹۸۷) و گروه همکاران). می توان اثبات کرد که در صورت تعدد گره هایی که در اپیدمیک ها مشارکت می کنند، کسر  $\alpha$  از گره ها که از قافله به روزآوری عقب

<sup>638</sup> round

<sup>639</sup> rumor spreading

<sup>640</sup> gossiping

یا به بیان علمی، در معرض باقی مانده ، در معادله زیر صدق می کنند:

$$s = e^{-(k+1)(1-s)}$$

شکل ۴-۳۲ ،  $\ln(s)$  را بعنوان تابعی از  $k$  نمایش می دهد. بعنوان مثال، اگر  $\ln(s) = -4.97$  و  $k=4$  ، در آنصورت ۶کمتر از ۰،۰۰۷ خواهد بود ، به این معنا که کمتر از ۷٪ ۰ گره ها در معرض باقی می مانند. در هر صورت، برای تضمین به روزآوری قطعی این گره ها هم نیازمند مقیاس های خاصی هستیم. ترکیب دو حالت ضد آنتروپی و شایعه پراکنی این آرزو را محقق می کند.

شکل ۴-۳۲- رابطه بین کسره گره های به روزآوری نشده و پارامتر  $k$  در شایعه پراکنی خالص. نمودار  $\ln(s)$  را به صورت تابعی از  $k$  نمایش می دهد.

یکی از مزایای عده الگوریتم های اپیدمیک ، مقیاس پذیری<sup>۶۴۱</sup> آنهاست. این ویژگی ناشی از این واقعیت است که تعداد همگامی های بین فرآیندها در قیاس با دیگر روش‌های انتشار نسبتاً ناچیز است. (۱۹۹۹) Marzullo and Lin نشان داده اند که بهتر است در سیستمهای حوزه گسترده، توپولوژی واقعی شبکه را در نظرگرفت تا کسب نتایج بهتر تضمین شود. در روش پیشنهادی آنها ، گره هایی که به تعداد بسیار محدودی از گره های دیگر متصل هستند، به احتمال نسبتاً زیادی تماس گرفته می شوند. فرض زیربنایی آن است که چنین گره هایی پل ارتباط با دیگر اجزای دور شبکه هستند؛ بنابراین، باید در اولین فرست ممکنه با آنها تماس ایجاد نمود. این روش اصطلاحاً به نام شایعه پراکنی جهت دار<sup>۶۴۲</sup> خوانده شده و انواع مختلفی دارد.

مشکل فوق اشاره به فرض مهمی دارد که در اغلب راه حل های اپیدمیک صدق می کند، به این صورت که یک گره می تواند به صورت تصادفی گره دیگری را برای پراکندن شایعه به آن انتخاب کند . به این معنا که بطور کلی، کل مجموعه گره ها باید برای تمام اعضا شناخته شده باشد. در سیستم های بزرگ، این فرضیه هرگز صدق نمی کند.

خوبیخانه، لزومی به دراختیار داشتن چنین لیست کاملی از اعضا وجود ندارد. همانطور که در فصل ۲ هم گفتیم، داشتن نمای تقریبی که کمابیش دائماً به روزآوری می شود، باعث سازماندهی مجموعه گره ها در یک گراف تصادفی خواهد شد. با به روزآوری مرتب نمای تقریبی هر گره، انتخاب تصادفی دیگر تبدیل به امری ساده خواهد شد.

## حذف داده ها

الگوریتم های اپیدمیک برای انتشار به روزآوری ها بسیار مناسب هستند. با این وجود، اثر جانبی عجیبی هم دارند: انتشار حذف یک آیتم داده ای مشکل می شود. این اشکال ناشی از این واقعیت است که حذف یک آیتم داده ای باعث تخریب تمامی اطلاعات روی آن آیتم می شود. در نتیجه، با حذف یک آیتم داده ای از یک گره، گره نهایتاً کپی های قدیمی از آیتم داده ای را دریافت کرده و آنها

<sup>641</sup> scalability

<sup>642</sup> directional gossiping

را بعنوان به روزآوری روی چیزی که قبل از نداشته تفسیر خواهد کرد.

ترفند این است که حذف یک آیتم داده ای را فقط بعنوان به یک روزآوری دیگر ثبت نموده و گزارشی از این حذف نگهداری کرد. به این ترتیب، کپی های قدیمی جدید و تازه وارد محسوب نشده و صرفاً بعنوان نسخه هایی تلقی می شوند که بوسیله عملیات حذف به روزآوری شده اند. ثبت حذف توسط انتشار گواهی های فوت<sup>۶۴۲</sup> انجام می شود.

البته مشکل گواهی های فوت آن است که باید در نهایت پاک شوند و در غیراینصورت، هر گره تدریجاً پایگاه داده ای محلی عظیمی از اطلاعات تاریخی درمورد آیتمهای داده ای حذف شده ایجاد خواهد کرد که دیگر قابل استفاده نخواهند بود. (Demers ۱۹۸۷) و گروه همکاران پیشنهاد می کنند که از چیزی به نام گواهی های فوت خاموش<sup>۶۴۳</sup> استفاده شود. تمامی گواهی های فوت پس از ایجاد مهرزمانی می خورند. اگر بتوان تصور کرد که به روزآوری ها تحت مدت زمانی محدودی به همه گره ها پراکنده می شوند، آنگاه می توان گواهی های فوت را پس از گذشت مدت زمان حداقل حذف نمود.

با این وجود، برای تهیه تضمین های محکم درمورد پراکنده شدن واقعی حذف شده ها به همه گره ها، تعداد بسیار محدودی از گره ها گواهی فوت خاموش دارند که هرگز دور انداخته نمی شوند. فرض کنید که گره  $P$  به ازای آیتم داده ای  $x$  چنین گواهی دارد. چنانچه تحت هر احتمالی به روزآوری قدیمی  $x$  به  $P$  برسد، آنگاه  $P$  صرفاً با پراکنده کردن گواهی فوت مجدد به ازای  $x$  واکنش خواهد داد.

### کاربردها

به عنوان خاتمه بحث بیایید نگاهی به برخی از کاربردهای جالب پروتکل های اپیدمیک بیندازیم. تا به اینجا انتشار به روزآوری ها را ذکر کردیم که احتمالاً پرکاربردترین موارد کاربرد محسوب می شود. همچنین، در فصل ۲ شرح دادیم که چگونه ارائه اطلاعات مکان یابی راجع به گره ها می تواند در ایجاد توپولوژی های خاص کمک کند. در همین راستا، از شایعه پراکنی می توان برای کشف گره هایی استفاده نمود که چند پیوند خروجی حوزه وسیع دارند، و سپس از شایعه پراکنی جهتدار به صورتی که در بالا گفته شد، استفاده کرد.

یکی دیگر از موارد مهم کاربرد پروتکل های اپیدمیک، جمع آوری یا درواقع، ابیاشتن اطلاعات است (Jelassi et al., ۲۰۰۵). حالت فرضی تبادل اطلاعات زیر را درنظر بگیرید. هر گره مفروض  $i$  ابتدا یک عدد اختیاری مثل  $x_i$  را انتخاب می کند. هنگامیکه گره  $i$  با گره  $j$  تماس برقرار می کند، هریک مقدار خود را به صورت زیر به روزآوری می کند:

$$x_i, x_j \leftarrow (x_i + x_j) / 2$$

مشخص است که پس از این تبادل،  $i$  و  $j$  هردو مقدار واحدی خواهند داشت. در واقع، به راحتی می توان مشاهده کرد که در نهایت تمامی گره ها یک مقدار واحد یعنی میانگین تمامی مقادیر اولیه را خواهند داشت. سرعت انتشار در اینجا هم نمایی است.

<sup>643</sup> death certificates

<sup>644</sup> dormant death certificate

محاسبه میانگین چه کاربردی دارد؟ حالی را در نظر بگیرید که در آن تمامی گره های  $i$ ،  $x_i$  را برابر با صفر قرار داده اند، به استثنای  $x_j$  که برابر با یک است.

$$\begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases}$$

اگر  $N$  گره وجود داشته باشد، آنگاه نهایتاً میانگین هر گره بصورت  $I/N$  محاسبه خواهد شد. در نتیجه، هر گره  $i$  می تواند ابعاد سیستم را براساس رابطه  $x_i = I/N$  بدست آورد. از این اطلاعات می توان به تنها ی جهت تنظیم پویای پارامترهای مختلف سیستم استفاده نمود. بعنوان مثال، ابعاد نمای تقریبی<sup>645</sup> (یعنی تعداد همسایگانی که هر گره های مشارکت کننده خواهد داشت. اطلاع از بستگی به تعداد کل گره های مشارکت کننده خواهد داشت. این رقم موجب می شود تا یک گره بتواند به صورتی پویا ابعاد نمای تقریبی خود را تنظیم نماید. در واقع، این عامل را می توان بعنوان یکی از ویژگی های خودمدیریتی تلقی نمود.

محاسبه میانگین ممکن است مخصوصاً هنگامی مشکل شود که گره ها بطور منظم به سیستم پیوسته و آنرا ترک کنند. یک راه حل عملی برای این مشکل، معرفی نقاط عطف زمانی<sup>646</sup> است. اگر فرض کنیم گره شماره  $i$  ایستا باشد، هنگامیکه گره  $i$  برای اولین بار با یک نقطه عطف زمانی مواجه می شود، متغیر  $x_i$  خود را برابر با صفر قرار داده و مجددآ شروع به محاسبه میانگین خواهد کرد.

البته، نتایج دیگر را هم می توان محاسبه نمود. بعنوان مثال، به جای آنکه گره ثابت ( $x_i = 0$ ) کار محاسبه میانگین را آغاز کند، به راحتی می توان گره تصادفی را با استفاده از رابطه زیر انتخاب نمود. هرگره  $i$  در ابتدا  $x_i = 0$  روی عدد اختیاری از فاصله یکسان مثلاً [0, 1] قرار خواهد داد و همچنین آنرا بطور دائم بصورت  $m_i$  ذخیره می کند. در اثر تبادل بین گره های  $i$  و  $j$ ، هر یک مقادیر خود را بصورت زیر تغییر می دهد:

$$x_i, x_j \leftarrow \max(x_i, x_j)$$

هر گره  $i$  که برای آن  $m_i < x_i$  باشد، در رقابت برسر آغازگر بودن جهت شروع محاسبه میانگین بازنده خواهد بود. در پایان، فقط یک برنده وجود خواهد داشت. البته، هرچند به راحتی می توان راجع به بازنده شدن یک گره نتیجه گیری کرد، اما تصمیم گیری درباره برنده شدن آن بسیار دشوارتر است، زیرا نمی توان مطمئن بود که تمامی نتایج وارد شده اند. راه حل این است که خوش بین باشیم: یعنی یک گره همیشه خود را برنده بداند مگر اینکه خلاف آن ثابت شود. در اینحال فقط باید متغیری را که برای محاسبه میانگین استفاده نموده، مجددآ روی صفر تنظیم کند. توجه داشته باشید که

<sup>645</sup> partial view

<sup>646</sup> epoch

ممکن است تا به اینجا محاسبات مختلف و متعددی (که در مثال ما محاسبه حد اکثر و محاسبه میانگین است) به طور همروند درحال اجرا باشند.

#### ۴-۶ - خلاصه

در اختیار داشتن امکانات قدرتمند و انعطاف پذیر برای ارتباطات بین فرآیندها برای تمامی سیستم‌های توزیعی نوعی ضرورت محسوب می‌شود. در کاربردهای متعارف شبکه، ارتباطات غالباً بر اساس عملیات اولیه سطح پایین عبور پیام استوار است که بواسیله لایه انتقال ارائه می‌شوند. یکی از مسائل مهم در سیستم‌های میان افزاری، پیشنهاد سطح بالاتری از انتزاع است. این امر باعث می‌شود تا بیان ارتباطات بین فرآیندها در مقایسه با پشتیبانی‌های ارائه شده برای رابط لایه انتقال آسان‌تر شود.

یکی از پرکاربردترین انتزاعات مورد استفاده، فراخوانی روال از راه دور (RPC) است. RPC بطور خلاصه به این معناست که یک سرویس بواسیله روالی پیاده سازی می‌شود که بدنه آن در سرور اجرا می‌شود. فقط امضای روال یعنی نام روال به همراه پaramترهای آن، به مشتری ارائه می‌شود. هنگامی که مشتری روال را فراخوانی می‌کند، پیاده سازی طرف مشتری که مجازی نامیده می‌شود، اقدام به بسته بندی مقادیر پaramتر به شکل یک پیام و ارسال آن به سرور می‌کند. این سرور، روال واقعی را فراخوانی کرده و نتایج را مجدداً به شکل پیام بازمی‌گرداند. مجازی مشتری مقادیر نتیجه را از پیام بازگشتی استخراج کرده و آنرا به برنامه کاربردی مشتری در حال فراخوانی باز می‌گرداند.

RPC‌ها امکانات ارتباطی همگام را ارائه می‌کنند که بواسیله آن، مشتری تا زمان ارسال پاسخ بواسیله سرور مسدود می‌شود. هرچند انواعی از مکانیزم‌ها وجود دارند که باعث انعطاف پذیری این مدل می‌شوند، می‌توان نتیجه گرفت که مدل‌های عام منظوره سطح بالای پیام گرا غالباً مناسب‌تر هستند.

در مدل‌های پیام گرا، مسئله مهم پایداری یا عدم پایداری ارتباطات و همچنین همگام و ناهمگام بودن آنهاست. بطور خلاصه، ارتباط پایدار به این معناست که پیامی که برای مخابره تحويل داده شده، تا زمان تحويل بواسیله سیستم ارتباطی ذخیره خواهد شد. به بیان دیگر، برای موقیت کار تبادل، لازم نیست که گیرنده و فرستنده سرپا و در حال اجرا باشند. در ارتباط ناپایدار، هیچ نوع امکانات ذخیره سازی ارائه نمی‌شود و گیرنده باید آمادگی داشته باشد تا پیام‌ها را به محفوظ ارسال آنها بپذیرد.

در ارتباط ناهمگام، به فرستنده امکان داده می‌شود تا فوراً پس از تحويل پیام برای ارسال، و احتمالاً حتی پیش از ارسال آن، به کار ادامه دهد. در حالی که در ارتباط همگام، فرستنده حداقل تا زمان دریافت پیام، مسدود می‌شود. در روش دیگر، فرستنده ممکن است تا زمان وقوع تحويل پیام و یا حتی تازمانیکه گیرنده، مشابه RPC‌ها، واکنش نشان دهد، مسدود شود.

مدل‌های میان افزاری پیام گرا غالباً ارتباط ناهمگام پایدار ارائه داده و در مواردی استفاده دارند که RPC‌ها دیگر مفید نباشند. از این مدل‌ها غالباً جهت کمک به تلفیق مجموعه‌های (بسیار پراکنده) پایگاه‌های داده ای به شکل سیستم‌های اطلاعاتی

پردامنه استفاده می شود. کاربردهای دیگر مدل های میان افزاری شامل پست الکترونیک و جریان کار می باشد.

یکی از اشکال بسیار متفاوت ارتباطات ، جریان سازی است که در آن مسئله این است که آیا دو پیام متولی رابطه زمانی دارند یا نه. در جریانات داده ای پیوسته ، به ازای هر پیام یک تأخیر انتها به انتهای حداکثر تعریف می شود. بعلاوه ، لازم است که پیامها براساس حداقل تأخیر انتها به انتها ارسال شوند. از نمونه های این نوع جریانات داده ای پیوسته می توان به جریانات تصویری و صوتی اشاره کرد. تعیین دقیق روابط زمانی یا انتظاری که از زیرسیستم ارتباطات زیرین از نظر کیفیت خدمات می رود، غالباً امری مشکل است. یکی از فاکتورهای پیچیده نقش لرزش است. حتی در صورت قابل قبول بودن میانگین کارآیی هم، تغییرات در زمان تحويل ممکن است منجر به غیرقابل قبول شدن کارآیی شود.

در نهایت اینکه ، یکی از مهم ترین انواع پروتکل های ارتباطی در سیستم های توزیعی، چندپیخشی است. در این پروتکل ها، ایده اصلی انتشار اطلاعات از یک فرستنده به چندین گیرنده است. راجع به دو پروتکل مختلف بحث نمودیم. در حالت اول، چندپیخشی با ایجاد یک درخت از فرستنده به گیرنده ها ایجاد شود. با توجه به دانش کامل فعلی ما در مورد نحوه سازماندهی در سیستم های نظیر به نظری ، راه حل هایی برای ایجاد پویای درخت ها بصورت غیرمت مرکز ایجاد شده است.

یکی دیگر از انواع مهم راه حل های انتشار مربوط به پروتکل های اپیدمیک است. این پروتکل ها هرچند بظاهر بسیار ساده می نمایند، در واقع بسیار مقاوم هستند. پروتکل های اپیدمیک علاوه بر انتشار پیام ها، به نحوی مفید و مؤثر برای انباشت اطلاعات در یک سیستم توزیعی بزرگ هم مورد استفاده قرار گیرند.

### مسائل فصل

۱- در بسیاری از پروتکل های لایه ای، هر لایه سرپیام خاص خود را دارد. مسلماً بهتر است که به جای داشتن چندین سرپیام جداگانه، یک سرپیام واحد در هر پیام قرار داده شده و تمام کنترل ها داخل آن قرار گیرد. چرا اینطور نیست؟

۲- چرا سرویس های ارتباط لایه انتقال معمولاً برای ساختن برنامه های کاربردی توزیع شده نامناسب هستند؟

۳- یک سرویس چندپیخشی مطمئن به فرستنده امکان می دهد تا با اطمینان پیام مورد نظر را به مجموعه ای از گیرنده ها انتقال دهد. آیا چنین سیستمی متعلق به لایه میان افزاری است یا باید بخشی از یک لایه سطح پایین تر تلقی شود؟

۴- روال *incr* را با دو پارامتر عدد صحیح در نظر بگیرید. این روال باعث اضافه شدن عدد یک به هر پارامتر می شود. حال فرض کنید که با یک متغیر واحد (*i,i*) دو مرتبه فراخوانی شود. اگر *n* در ابتداء صفر باشد، پس از استفاده از فراخوانی توسط مرجع چه مقداری خواهد داشت؟ همین مسئله را با استفاده از کپی/بازیابی حل کنید.

۵- در C ساختاری به نام union وجود دارد که طی آن فیلد یک رکورد ( که در C struct نامیده می شود) می تواند هریک از چندین مقدار مختلف را در خود نگه دارد. در زمان اجرا ، بطور

- قطع نمی توان تعیین کرد که کدامیک موجود است . آیا این ویژگی C اثراتی بر روال فرآخوانی از راه دور دارد؟ توضیح دهید.
- ۶- یک روش برای انجام تبدیلات پارامتر در سیستمهای RPC آن است که هر ماشین پارامترها را در قالب ارائه داخلی خود ارسال نموده و ماشین دیگر در صورت نیاز ، کار تبدیل را انجام دهد. سیستم داخلی را می توان به کمک یک کد در بایت اول بیان کرد. با این وجود، از آنجاییکه مکان یابی بایت اول در کلمه اول مشکل اصلی است ، آیا این روش کار می کند؟
- ۷- فرض کنید که یکی از مشتری ها ، RPC ناهمگامی به یک سرور را فرآخوانی می کند و سپس منتظر باقی می ماند تا سرور با استفاده از RPC غیرهمزمان دیگر نتیجه را عودت دهد. آیا این روش مشابه آن است که به مشتری اجازه دهیم یک RPC عادی را اجرا کند؟ در صورت تعویض RPC های ناهمگام با RPC های همگام چه اتفاقی خواهد افتاد؟
- ۸- به جای اینکه به سرور اجازه دهیم که خود را در محافظی ثبت کند، مانند آنچه در DCE انجام می شود ، می توان همیشه یک نقطه پایانی واحد را به آن نسبت داد. سپس از این نقطه پایانی می توان برای ارجاع به اشیاء در فضای آدرس سرور استفاده نمود. نقطه ضعف عمدۀ این طرح پیشنهادی چیست؟
- ۹- آیا تمایز قائل شدن بین RPC های ثابت و پویا مفید خواهد بود؟
- ۱۰- توضیح دهید که در صورت استفاده از سوکت ها، ارتباطات بدون اتصال در بین مشتری و سرور به چه صورت انجام خواهد شد؟
- ۱۱- تفاوت بین عملیات اولیه MPI\_isend و MPI\_bsend را توضیح دهید.
- ۱۲- فرض کنید که فقط از عملیات اولیه ارتباط ناهمگام ناپایدار شامل فقط یک عمل اولیه دریافت ناهمگام می توانید استفاده کنید. عمل اولیه مربوط به ارتباط همگام ناپایدار را چگونه پیاده سازی خواهید کرد؟
- ۱۳- فرض کنید که فقط از عملیات اولیه ارتباط همگام ناپایدار می توانید استفاده کنید. عمل اولیه مربوط به ارتباط ناهمگام ناپایدار را چگونه پیاده سازی خواهید کرد؟
- ۱۴- آیا اجرای ارتباط ناهمگام پایدار بواسیله RPC ها امکانپذیر هست؟
- ۱۵- در متن گفتم که برای شروع اتوماتیک فرآیندی برای آوردن پیام ها از یک صف ورودی، غالباً از محافظی استفاده می شود که بر صف ورودی نظارت می کند. اجرای متفاوتی را ذکر کنید که در آن از محافظ استفاده نشود.
- ۱۶- جداول مسیردهی در IBM WebSphere و همچنین در بسیاری از دیگر سیستم های صف بندي پیام، به صورت دستی پیکربندی می شوند. روش ساده ای را برای انجام اتوماتیک اینکار ذکر کنید.
- ۱۷- در ارتباط پایدار، گیرنده غالباً دارای بافر محلی است که می توان در صورتی که گیرنده در حال اجرا نباشد، پیام ها را در آن ذخیره نمود. برای ایجاد چنین بافری ، باید حتماً ابعاد آنرا هم تعیین و تخصیص نمود. درباره ترجیح دادن این روش و همچنین برعلیه مشخص ساختن اندازه بافر بحث کنید .

- ۱۸- توضیح دهید که چرا ارتباط همگام ناپایدار دارای مشکلات ذاتی مقیاس پذیری است و چه راه حلی برای آنها پیشنهاد می کنید؟
- ۱۹- نمونه ای ذکر کنید که در آن، چندپخشی برای پخش جریانات داده ای گستته هم مفید و مورد استفاده باشد.
- ۲۰- فرض کنید که در یک شبکه حسگر ، درجه حرارت های اندازه گیری شده بوسیله حسگر مهر زمانی ندارند، اما فوراً به اپراتور ارسال می شوند. آیا صرف تضمین تأخیر انتها به انتهای حد اکثر کافی خواهد بود؟
- ۲۱- در صورتی که مجموعه ای از کامپیوترها به شکل یک حلقه ( منطقی یا فیزیکی) سازماندهی شده باشند، چگونه می توان تأخیر انتها به انتهای حد اکثر را تضمین نمود؟
- ۲۲- در صورتی که مجموعه ای از کامپیوترها به شکل یک حلقه ( منطقی یا فیزیکی) سازماندهی شده باشند، چگونه می توان تأخیر سربه سری حداقل را تضمین نمود؟
- ۲۳- با وجودی که چندپخشی از نظر فنی قابل اجراست ، حمایت چندانی از اجرای آن در اینترنت صورت نگرفته است. راه حل این مشکل در مدلهای شغلی حقیقی یافته می شود. یعنی اینکه هیج کس نمی داند چگونه می توان از چندپخشی بعنوان وسیله ای برای کسب درآمد استفاده کرد. چه طرحی پیشنهاد می کنید؟
- ۲۴- معمولاً، درخت های چندپخشی سطح برنامه کاربردی با توجه به کشیدگی بهینه سازی می شوند که براساس مقدار تأخیرها و تعداد پرشها اندازه گیری می شود. موردي را ذکر کنید که استفاده از این معیار سنجش می تواند باعث ایجاد درخت های بسیار نامناسب شود.
- ۲۵- در هنگام جستجو برای فایل ها در یک سیستم نظیر به نظر ساخت نیافته ، ممکن است محدودسازی جستجو به گره های دارای فایل های مشابه فایل های مورد نظر باعث تسهیل کار شود. توضیح دهید که شایعه پراکنی چگونه می تواند به پیدا کردن این گره ها کمک کند؟