ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

به نام خداوند مهربان

تمرین شماره اول سیستم عامل 2 محمد تنهایی 87202446 نرم افزار

فصل اول

1. در سیستم های توزیع شده شفافیت به این معنی است که کاربر سیستم (کاربر عادی یا برنامه نویس) متوجه نشود که با یک سیستم توزیع شده مواجه است ، به عبارت بهتر کاربر حس کند که با یک سیستم یک پارچه مواجه است . این شفافیت در سطوح مختلف مطرح می شود . به عنوان مثال شفافیت در سطح مکانی یعنی اینکه کاربر نتواند متوجه شود که داده ها در کجا قرار دارند (به عنوان نمونه بخشی از داده های دانشگاه در دانشکده کامپیوتر و بخشی دیگر در دانشکده فیزیک قرار داشته باشند ولی کاربر چنین چیزی را حس نکند) . در سطح انتقال یعنی داده های این آزادی را داشته باشند که با حفظ نام خود از یک مکان به مکان دیگر منتقل شوند (مثلاً داده هایی که در سرور مهندسی کامپیوتر با نام usr/people/ قابل دسترسی هستند اگر از سرور کامپیوتر به سرور فیزیک منتقل شود تغییری در آن بوجود نیاید ، به عبارت بهتر کاربر متوجه این تغییرات نشود) ، در سطح Relocation یعنی اینکه کاربر متوجه انتقال داده های در حین استفاده از یک مکان به مکان دیگر نشود (به عنوان نمونه ممکن است که داده ها را بین دو یا چند سرور جابه جا کنیم). در سطح تکرار یعنی اینکه کاربر متوجه نشود در مکان های مختلفی داده ها تکرار شده اند (مثلاً می توانیم داده ها را در مکان های مختلفی تکرار کنیم. در مثال ما بر فرض ، داده ها به صورت مشابه در سرور کامپیوتر و فیزیک تکرار شده اند اما کاربر این واقعیت را متوجه نمی شود ، معمولاً تکرار برای افزایش میزان دسترسی پذیری و سرعت آن انجام می شود) . در سطح Concurrency یعنی مخفی کردن رقابت کاربران مختلف برای به دست آوردن منابع ، به عبارت بهتر کاربران متوجه وجود کاربران دیگر در سیستم نشوند (مثلاً اگر 2 یا چند کاربر با هم داخل سیستم شدند سیستم با زمانی که یک کاربر در سیستم وجود دارد تفاوتی نکند) .شفافیت در سطح موازات یعنی برنامه بتواند به صورت موازی در چند پردازنده اجرا شود و نیازی به دانستن این مسئله توسط برنامه نویس نباشد . بدیهی است که سخترین شفافیت ممکن است. (مثلاً برنامه ای برای دانشگاه نوشته شود که روی یک پردازنده تک پردازنده ای کار می کند این برنامه بتواند به صورت موازی روی شبکه اجرا شود و نیازی به برنامه نویسی موازی از طرف برنامه نویس نباشد). شفافیت در سطوح دیگری نظیر access ، Persistence ، Failure و ... نیز مطرح می شود . در عمل شفافیت را در سطح کامل آن پیاده سازی نمی کنیم ، هر چه شفافیت را بالاتر ببریم کارایی پایین تر می آید. به همین سبب است که در بسیاری از کاربردها کارایی سیستم را فدای شفافیت نمی کنیم . و معمولاً شفافیت را در حد قابل قبولی (نه لزوماً 100 درصد) نگه می داریم.
2. مقیاس پذیری به معنی این است که سیستم چقدر قابلیت توسعه دارد . مقیاس پذیری در چند زمینه مطرح می شود . اندازه : یعنی این که بتوانیم کاربران و منابع جدیدی به سیستم اضافه کنیم ، Geography کاربران و منابع می توانند دور از هم باشند و در سطح مدیریت یعنی این که مدیران مختلفی در سیستم بتوانند مدیریت کنند. برای افزایش توسعه پذیری باید از جمع کردن اطلاعات و امکانات در یک نقطه پرهیز کنیم . معمولاً سیستم های توسعه پذیر علاوه بر سیستم های نرم افزاری قابل توسعه ، دارای سخت افزار قابل گسترش نیز هستند . در نظر بگیرید که اگر سیستم اینترنت در یک مرکز خاص تجمیع می شد چه مشکلات عدیده ای بوجود می آمد . در حالیکه با طراحی حاضر (یعنی توزیع شده) مشکلات آن برای توسعه بسیار کمتر است .
تکنیک های توسعه پذیری شامل : توزیع (در بالا توضیح داده شد) ، تکرار و caching است.

ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

3. بدترین Delay زمانی اتفاق می افتد که بخواهیم از گوشه سمت چپ بالا به گوشه سمت راست پایین بیاییم . این امکان وجود دارد در این حالت که بدترین حالت ممکن است باید 15 hop را به صورت افقی و 15 hop را به صورت عمودی برویم کـه مجمـوع 30 تاخیر به دست می آید.

4. In this case we have : $\left(\frac{1}{4}\right)^n < \frac{1}{100} \rightarrow 4^n > 100 \rightarrow n \geq 4$

5. چیزی در این جا واضح است این است که P0 منتظر P1 می ماند ، P1 منتظر P2 است و الی آخر . تا اینکه به آخر زنجیر برسیم . بالاخره Pn جواب Pn-1 را داده و به صورت معکوس همه reply ها دریافت می شوند . کارایی P0 در بدترین حالت خود قرار دارد . در این حالت P0 باید منتظر بماند که اجرای همه پردازه های این Chain اجرا شوند! هر چه پردازه به اول لیست (i کمتـر باشـد) نزدیک تر باشد کارایی پایین تری دارد . در این حالت تاخیر P0 نیز مـاکزیمم مقـدار خـود اسـت. (بـه انـدازه n-2 بایـد reply و request انجام شود) . حالت بد دیگر رفتار غیر معمول از یک پردازه است که ممکن است کل سیستم را دچار اختلال کند.

6. به طور واضح دسترسی پذیری سیستم replicate بالاتر از سیستم partitioning است .(در سیستم توزیع شـده partitioning به دلیل پخش بودن داده ها [و احتمالا دور بودن سرور ها از هم] دسترسی پذیری پایین تر از زمانی است که همه سرور هـا یـک کپی از داده ها را در خود نگهداری می کنند.)

از طرفی در سیستم replicate قابلیت اعتماد پایین تر از سیسـتم هـای partitioning اسـت . مشـکل اصـلی در سیسـتم هـای replicate هماهنگ و سازگار بودن داده های سرور های مختلف است و ممکن است که داده ها در سرور های مختلف بـا یکـدیگر متفاوت باشند! در حالیکه با توجه به ماهیت سیستم partitioning چنین چیزی در این سیستم وجود ندارد و قابلیـت اعتمـاد آن بالاتر است.

از لحاظ استفاده بهینه از فضا در سیستم replicate اتلاف منابع داریم . زیرا یک داده چندین بار تکرار مـی شـود . در حالیکـه در سیستم partitioning بالاترین استفاده از منابع را داریم . در عمل استفاده از hybrid ای از آنها مناسب تر است .

از لحاظ گسترش پذیری و مقیاس پذیری سیستم های partitioning بهتر عمل می کنند . ممکن اسـت داده هـا آن قـدر بـزرگ شوند که امکان ذخیره آن ها در یک سرور وجود نداشته باشد در این حالت سیستم replicate تعطیل می شود! و تنها گزینه بـاقی مانده استفاده از سیستم های partitioning و یا مخلوطی از آنهاست! (یعنی دیتا سنترهایی به وجود آوریم که هر کدام در داخـل دیتا سنتر از سیستم partitioning استفاده کنند ولی هر دیتا سنتر داده ها را replicate کند).

جنبه دیگر توسعه پذیری بحث های جغرافیایی است . به عنوان مثال نگهداری داده های DNS هر کشور در داخـل آن کشـور کـار توسعه آن را ، بسیار آسان تر از این می کند که داده ها همگی در یک سیستم جهانی نگهداری شوند . در این حالت نـه تنهـا بحـث شیوه گسترش و مدیریت آن دچار مشکل می شود ؛ حتی پای مسائل پیش بینی نشده مانند سیاست و غیره به این سیستم هـا بـاز می شود .

مقایسه این 2 سیستم از لحاظ کارایی اندکی مشکل است . ابتدا باید تعریف خود را از کارایی مشخص کنیم . ممکـن اسـت میـزان استفاده از CPU مقیاس کارایی باشد در این صورت به نظر می رسد که در سیستم های partitioning استفاده از CPU در همه سرور ها به اندازه هم باشد (به شرطی که داده ها به صورت تصادفی مورد دسترسی قرار گیرنـد) امـا در سیسـتم هـای replicate ممکن است که یک سرور بسیار فعال باشد (و نتواند جواب درخواست ها بدهد) و یک سرور بی کار (به نوعی می توان گفت کارایی پایین آمده است). از طرف دیگر اگر بحث های تاخیر دریافت و ارتباط را به میان بیاوریم متوجه می شویم که سیسـتم replicate بهتر از partitioning عمل می کند . زیرا ممکن است بر حسب نوع توزیع در سیستم های partitioning منـابع قابـل دسـتیابی بسیار از کاربر دور باشند در این حالت است که کارایی سیستم پایین می آید .

به صورت خالص هیچ کدام از این دو سیستم پیشنهاد نمی شود . میکسی از این 2 مناسب به نظر می رسد.

ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	**Multicomp.**		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	Varies
Openness	Closed	Closed	Open	Open
In the system, is it fundamental to have a shared clock?	بله	بله	نه	بله
Is a shared time essential? How accurate is should be?	بله	نه ، ولی می توان چنین ساعتی داشت	نه	نه
Is there a shared memory?	بله	نه	نه	نه
Are all decisions local? Distributed?	محلی ، توزیع شده	محلی ، توزیع شده	غیر محلی ، غیر توزیع شده	تا حدودی محلی ، توزیع شده
How fast is the communication channel? How error-prone?	سریع ، خطا نادر	متوسط ، امکان خطا زیاد	تا حدودی کند ، امکان خطا بسیار زیاد	تا حدودی کند ، امکان خطا بسیار زیاد

ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

فصل دوم

1. transport-level communication به سختی می تواند شفافیت را تامین نماید .در این نوع ارتباط برنامه نویس باید مراقب باشد که به کجا و چگونه ارتباطات برقرار می شود . به عبارت بهتر برنامه ها اختصاصی می شوند . این نوع ارتباط نمی تواند تامین کننده یک ارتباط درست برای ساخت برنامه های توزیع شده باشد . به عنوان مثال از این وضعیت برنامه های سوکتی نمی توانند ارتباط مناسبی را با برنامه های دیگر داشته باشند .
2. سیستم عامل اگر در زمان اجرا نوع متغیر را تشخیص ندهد نمی تواند به درستی آنرا پیمایش و اجرا کند . بنابراین این Union ها در سیستم RPC قابل استفاده نیستند مگر اینکه به وسیله تگ و یا هر وسیله مشخصی نوع آنها را معین کنیم. این مشخصه هم پس از تعیین قابل ویرایش توسط کاربر نخواهد بود .
3. یک پیاده سازی dynamic invocations می تواند تمام انواع پیاده سازی ها شامل استاتیک و دینامیک را پشتیبانی کند . به منظور سادگی می توان کلیه invocations به وسیله همین یک نوع پیاده سازی کرد . خوبی این کار این است که تنها یک نوع implementation داریم و بدی آن نیز پایین آمدن کارایی است . و ممکن است از راه حل های بهینه فاصله بگیریم . در واقع راحتی خودمان را فدای راحتی سیستم کرده ایم!!
4. تعیین اندازه بافر و مکان آن توسط کاربر پیاده سازی سیستم بافرینگ را راحت تر می کند . هر چند که اگر بافر سر ریز داشته باشد داده ها گم می شوند . اما کار برنامه نویس راحت می شود و نیازی به برنامه نویسی ندارد .

   در نقطه مقابل برنامه نویسی و تشخیص اتوماتیک بافر داریم که با تکنیک هایی مانند بافر دهی نمایی (دو برابر کردن بافر در صورت نیاز) و کارهایی از این قبیل قابل انجام است . اما به هر حال نیاز به کار بیشتر برنامه نویس است .
5. این کار قابل انجام است . فرض کنیم که پیام ها را دریافت می کنیم و برای گم نشدن در بافری مخصوص این کار نگهداری می کنیم . تا به حال دریافت نا همگام داشته ایم. اما تحویل پیام ها را به صورت همگام پیاده سازی می کنیم . به این صورت که تا از سیستم عامل یک پالس (پیام یا هر چیزی) مبنی بر تحویل پیغام دریافت نکردیم پیغام را تحویل نخواهیم داد!! این پالس ها می توانند به صورت یک کلاک پیاده سازی شوند .(حتی این کلاک می تواند با متد هایی در شبکه هماهنگ شود) .پس تحویل همگام خواهیم داشت.

حالت در سیستم های توزیع شده

1. **الف)** به این معنی است که : اگر یک سرور پیغامی دریافت کرد حتماً قبل از دریافت این پیغام روی برد مبدا نوشته شده است . به عبارت بهتر امکان ندارد که سرور یک پیغام دریافت کند که هیچ کس آنرا نفرستاده باشد.

   ب) در این سیستم تاخیر رسیدن پیغام ها برابر نیست . بنابر این ممکن است که سناریویی شبیه زیر داشته باشیم :

   A به C پیغام m را می فرستد ، پس از این زمان پیغام m’ را می فرستد . ممکن است که پیغام m’ حاوی اطلاعاتی در مورد لغو عمل درخواستی m باشد! رسیدن این پیغام قبل از m می تواند باعث اخلال سیستم شود. به عبارت بهتر در یک سیستم واقعی علاوه بر اینکه نیاز به رعایت ترتیب علی داریم باید ترتیب دریافت نیز یکی باشد . سناریوی دیگر این است که هر پردازه به ترتیب متفاوت از بقیه پیغام ها را دریافت کند . به عنوان مثال یک پیغام سراسری m و سپس پیغام m’ ارسال می شود . حال اگر برخی این دو را mm’ و برخی m’m دریافت کنند مشکلات بیشتری برای سیستم به وجود می آید . بر این اساس ما باید چند قاعده به سیستم اضافه کنیم .

با وجود چنین فرضیاتی ، جالب است بدانیم که ، وجود ترتیب علی برای طراحی سیستمی که مشکلات بالا را نداشته باشد کافی است . در واقع بدبینانه ترین فرض ممکن برای سیستم این است که در سیستم توزیع شده تنها ترتیب علی را داشته باشیم . چنین فرضی در دنیای واقعی منطقی است . زیرا امکان ندارد معلولی بدون علت داشته باشیم.

ج) می‌توان سوابق علی را مرتب کوتاه نمود تا مناسب برای ساعت باشند. یعنی بخشی از سوابق که برای همه وقایع مشترک است، حذف شوند. بنابراین سابقه علی می‌تواند با یک آرایه ثابت (اندازه بعدها) ارائه شود.

تصویر $\theta(e)$ روی پردازه P_iهمانا یک پیشوند از سابقه محلی P_i است.

$$\theta_i(e) = h_i^k \quad \text{for some k and} \quad e_i^l \in \theta_i(e) \quad \text{for all } l<k$$

بنابراین یک عدد طبیعی کافی است تا مجموعه $\theta_i(e)$ را نشان دهد.

چون $\theta(e)= \theta_1(e) \cup \theta_2(e) \cup \ldots \cup \theta_n(e)$ پس کل سابقه علی می‌تواند با یک بردار n بعدی از اعداد طبیعی نشان داده شود: $VC(e)$ (یعنی بردار ساعت e) طوری که برای همه $1 \le i \le n$ داریم:

$$VC(e)[i] = K \text{ , iff } \theta_i(e) = h_i^K$$

بنابراین هر پردازه P_iیک بردار محلی از اعداد طبیعی (VC) نگهمیدارد که $VC(e_i)$ معروف به ساعت بردار P_i وقتی که e_i را اجرا کرده است، می‌باشد.

هر پیغام m شامل یک زمان‌مهر $TS(m)$ است که مقدار بردار ساعت واقعه ارسال m است.

قواعد بردار ساعت عبارتنداز:

$VC(e_i)[i] := VC[i] + 1$ if e_i is a <u>send</u> or an <u>internal</u> event.
$VC(e_i) := \max \{VC, TS(m)\}$ if e_i = receive (m)
$VC(e_i)[i] := VC[i] + 1$

فهم ما از j امین عنصر بردار ساعت پردازه P_i: (به ازای همه j: $i \neq j$)

$$VC(e_i)[j] := \text{\# of events of } P_j \text{ that casually precede } e_i \text{ of } P_i$$

$VC(e_i)[i]$ معرف تعداد وقایعی از P_i است که تا e_i (و شامل e_i) اجرا شده‌اند.

تعریف رابطه < بین دو بردار ساعت:

$$V < V' \equiv (V \neq V') \wedge (\forall k: 1 \le k \le n: V[k] \le V[k'])$$

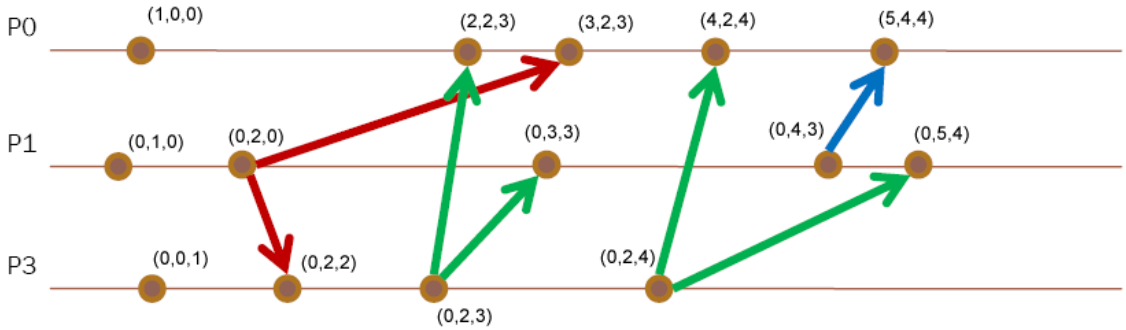
د) با توجه به بردار ساعت و روابط بین آنها داریم :

Assume that (4,5,6) indicate e2 , (4,5,8) e3 and (5,2,8) e1 so we can recognize the relation between events :

می توانیم بین e2 و e3 رابطه علی پیدا کنیم . این ارتباط می تواند خارجی باشد .به این صورت که e2 یک پیغام به e3 می دهد و e3 بر اساس بردار ساعت خود و بردار ساعت e2 اطلاع خود را به روز می کند . اما هیچ ارتباطی بین e3 و e1 قابل تشخیص نیست. زیرا e1 حتما باید از e3 پیام دریافت کند (زیرا اطلاع آن از پردازه سوم به اندازه اطلاع e3 است) ، با دریافت پیام از e3 بردار ساعت پردازه اول به روز می شود . بر این اساس اطلاع رخداد اول از پردازه دوم 5 می شود که این اطلاع با فرضیات مسئله (که اطلاع آن از پردازه دوم به میزان 2 است) در تناقض است! پس چنین ترتیب علی امکان پذیر نیست .

ABBYY PDF Transformer 2.0 Click here to buy www.ABBYY.com

2. الف) فرض کنیم که P0 هم جز حالت های سراسری باشد .(اگر P0 را حساب نکنیم بردار ما دارای 2 مولفه خواهد بود و مولفه اول را این بردار 3 مولفه ای را می توانیم در نظر نگیریم)



b)

$I. f \rightarrow j$

$II. a \,||m$

$III. c \rightarrow j$

$IV. k \rightarrow e$

$V. j \leftarrow k$