جلسه هفتم

کشف بن بست و ترمیم در سیستم های توزیع شده

کشف بن بست در الگوریتم متمرکز

در این روش یک پردازه هماهنگ کننده یا Cordinator وجود دارد که پردازه ها در خواست هایشان را به Cordinator می فرستند این پردازه هماهنگ کننده گراف تخصیص منابع را تشکیل داده و اگر بن بست رخ داده باشد متوجه می شود که پردازه ها در بن بست قرار دارند سپس مطابق روش هزینه کمینه یک یا تعدادی از پردازه ها را خاتمه می دهد.

معیار هزینه کمینه می تواند موارد زیر باشد

- چه مدتی از اجرای پردازه گذشته است
- چه مدتی از اجرای پردازه باقی مانده است
- چه منابعی و به چه تعداد در اختیار پردازه است
- تعداد فایل های باز شده
- میزان استفاده از حافظه

نکته: پردازه Cordinator در M.W قرار دارد.

مشکلات روش متمرکز:

1- نقطه خرابی دارد اگر پردازه هماهنگ کننده از بین برود دچار مشکل خواهد شد.

2- همه باید درخواست های خود را به Cordinator بفرستند که باعث ایجاد گلوگاه می شود.

کشف بن بست در الگوریتم توزیع شده

در این روش پردازه ای که به منبعی نیاز داشته باشد در خواستش را می فرستد، اگر منبع در اختیار پردازه دیگری باشد و این پردازه به منبع دیگر نیاز داشته باشد درخواستش را می فرستد، اگر به همین ترتیب آخرین پردازه درخواستش به پردازه شروع کننده برسد حلقه انتظار چرخشی به وجود آمده و پردازه های موجود در حلقه دچار بن بست شده اند. هر پردازه مدتی صبر می کند و در صورت نرسیدن منبع خودش را از بین می برد(Kill).

عیب این روش این است که ممکن است Over Kill رخ دهد، یعنی همه پردازه ها خود را از بین ببرند.

روش ostrich(شتر مرغ)

در این روش سیستم هیچ تمهیداتی جهت مقابله با بن بست در نظر نمی گیرد و اگر بن بست رخ دهد سیستم به صورت دستی Restart می شود و یا در صورت امکان منابعی را اضافه می کنیم.

تراکنش در سیستم های توزیع شده(Trasaction)

نقش کلیدی دارد. هر پردازه ای از طریق یک یا چند تراکنش اجرا می شود، جهت بر آورده شدن جامعیت سیستم، هر تراکنش می بایست چهار خاصیت زیر را داشته باشد(ACID):

Atomicity : یا همه دستورات تراکنش اجرا می شود با هیچ کدام

Consistency : دستورات هر تراکنش بایستی ساز گار باشند

Isolated : تراکنش ها اثر مخرب روی همدیگر نداشته باشند

Durable: قبل از اینکه Commit کنند باید نتیجه را در جائی ذخیره کنند.

پیاده سازی تراکنش ها

- به صورت محلی
- به صورت سراسری

پیاده سازی تراکنش ها به صورت محلی:

- PWS (Private wide Space)
- WaL (Write ahead Log)

نکته: محلی به این خاطر است هر کلاینت یا سرور حواسش به اجرای محلی نیز باشد، زیرا ممکن است در اجرای محلی نیز مشکل پیش بیاید.

روش PSW : هر پردازه ای در حافظه محلی اش فضائی را به اندازه فایل در نظر می گیرد و سپس فایل را در آن محل Load کرده و دستورات تراکنش بر روی آن اعمال شده و در انتهای فایل ذخیره می شود.

اشکال این روش این است که اگر اندازه فایل بزرگ باشد می بایست حافظه زیادی برای این کار در نظر گرفت.

روش WaL : در این روش دستورات تراکنش قبل از اجرا در یک فایل Log یا فایل ثبت ذخیره شده تا در صورت بروز خطا مانند قطع برق یـا قطع ارتباط بین دو ماشین و ... عمل Rule back و باز گرداندن به حالت اولیه انجام شود(مثال زیر را ببینید).

x = 0;	Log	Log	Log
y = 0;			
BEGIN_TRANSACTION;			
x = x + 1;	[x = 0 / 1]	[x = 0 / 1]	[x = 0 / 1]
y = y + 2		[y = 0/2]	[y = 0/2]
x = y * y;			[x = 1/4]
END_TRANSACTION;			
(a)	(b)	(c)	(d)

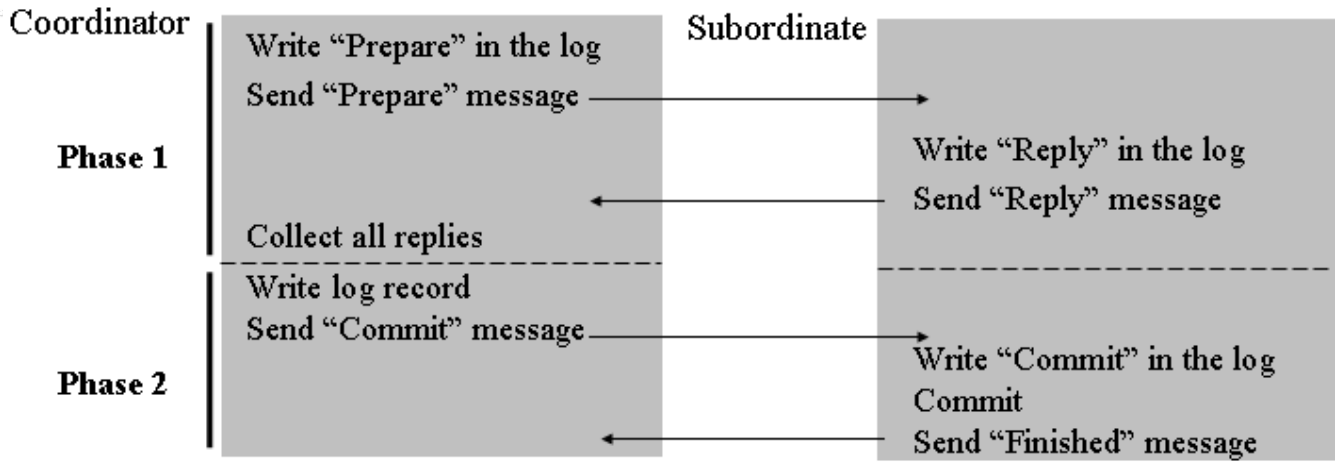
(a) : یک تراکنش.

(b) تا (d) : Log قبل از اجرای هر جمله.

نکته: با استفاده از Log هم می توان به جلو (do) رفت و هم می توان به عقب (undo) رفت.

پیاده سازی تراکنش به صورت سراسری(Two Logic Commit)

- عمل commit کردن یک تراکنش بایستی atomic صورت گیرد
- در یک سیستم توزیعی متشکل از چند پردازه موجود در چند ماشین، که هریک تعدادی متغیر فایل، DB و اشیائی دیگر را در خود جای داده اند، شاید همکاری پردازه ها برای دستیابی به این هدف لازم باشد.
- پردازه ای که تراکنش را اجرا می کند بعنوان Coordinator عمل کرده و آغاز عمل commit کردن را در حافظه پایدار ثبت کرده و پیغام هایی را مبنی بر این تصمیم به کلیه پردازه های زیر دست ارسال می کند.
- زیردستان در صورت آمادگی، خود را ثبت کرده و به Coordinator پاسخ مثبت می دهند.
- در صورت وصول پاسخ مثبت از کلیه زیر دستان Coordinator،تراکنش را commitاعلام و ثبت نموده، پیغام هایی را بر این مبنی به کلیه زیر دستان ارسال می کند.
- زیر دستان پس از دریافت پیغام های فوق، commit خود را ثبت کرده و پیغام اتمام کار را به Coordinator می فرستند.



نکات طراحی Client

- می بایست User Inter face(UI) خوبی دشته باشد.
- می بایست Trancparency داشته باشد تا کاربر متوجه توزیع شدگی نشود.
- Client می بایست عمل Cashing را انجام دهد تا کاربر متوجه Proxy نشود.

انواع Server

- سرور های تکرای
- سرور های همزمان

سرورهای تکراری : در سرور های تکراری سرور منتظر درخواست می باشد و به محض دریافت یک درخواست به درخواست جواب داده و بعد به سراغ درخواست های بعدی می رود به عبارتی از برنامه نویسی ایجاد tread استفاده نمی کند ولی در روش سرور های همزمان سرور برای هر درخواست یک tread ایجاد می کند به عبارتی مثل این که چند سرور داشته باشیم.

ارتباط Client با Server :

از طریق یک پورت این ارتباط میسر می شود، هر درخواست روی یک پورت ارسال می شود مثل FTP روی پورت 21، هر سروری بر روی پورت مخصوص خود منتظر درخواست است مثلا FTP Sever بر روی پورت 21 منتظر است تا به درخواست هایش پاسخ دهد.

نکته1: هر سرویس یا هر درخواست از طریق یک پورت خاص انجام می شود مانند FTP روی پورت 21 ، Email روی پورت 23 و HTTP روی پورت 80

نکته2: هر Client درخواستش را به صورت یک آدرس IP و یک متن بر روی Super Server ارسال می کند و Super Server با توجه به متن مربوطه سرور مربوطه را فراخوانی می کند.

نکته: روش اول در Windows کاربرد دارد و روش دوم Unix

روش های قطع ارتباط بین Client و Server

در نظر گرفتن time out : سرور یک تایمر را Set می کند و پس از مدتی (مشخص) اطلاعاتی از Client نیاید ارتباط را قطع می کند.

State Full : یعنی تمامی مکالمات بین کلاینت و سرور ذخیره شود.

State Less : لزومی به ذخیره مکالمات نیست

نکته: Upload کردن بهتر است State Full باشد ولی ULR بهتر است State Less باشد زیرا در این صورت می بایست اطلاعات زیادی را ذخیره کرد.

مهاجرت کد: کاربرد آن در پردازش موازی(parallel processing) است که یک برنامه بر روی چندین ماشین در حال اجرا است

مزایای مهاجرت کد:

- اجرای سریعتر: یک کار توسط چند نفر سریعتر انجام می شود.
- توازن بار(Load blancing): کار بین همه تقسیم می شود.
- انعطاف پذیری: که باعث سرعت اجرای بیشتر می شود.

مهاجرت کد به صورت پویا:

وقتی کلاینت درخواستی را روی سرور می دهد، سرور کد مورد نظر را به کلاینت ارسال کرده و بر روی کلاینت اجرا می شود به عنوان مثال مـی توان به اسکریپت های جاوا اشاره کرد، یعنی این که از سرور به کلاینت می رود و روی کلاینت اجرا می شود و یا مثلا سایت سنجش اگر در وارد مشخصات اشتباهی رخ دهد همان جا اعلان می شود.

قطعه های هر برنامه:

قطعه کد: کد برنامه اجرائی که به صورت دودوئی است

قطعه منابع: فایل ها و دیگر منابع مورد استفاده مثل فایل های DLL

قطعه اجرای: انباره، ثبات های سیستم و پشته و...، مثلا در زمان Context Swiching باید اطلاعات در جائی ذخیره شوند.

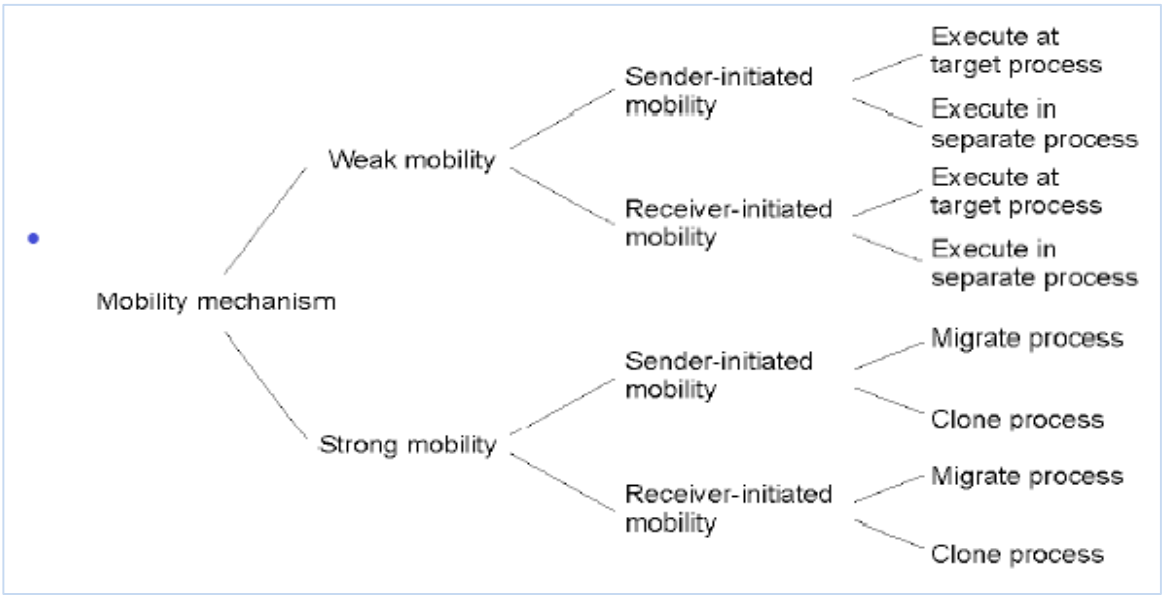
مهاجرت قطعه کد:

Weak Mobility : قطعه هائی که قبل از اجرا مهاجرت می شوند مثل اسکریپت های جاوا

Strong Mobility : در حین اجرا کد ارسال می شود به عنوان مثال در شبکه های حس گر

نکته: مهاجرت قوی به دلیل fail کردن سیستم می باشد

به طور کلی مدل های مهاجرت کد به صورت زیر می باشد.



Sender : یعنی کسی که کد را دارد یا کد بر روی آن در حال اجراست. درخواست مهاجرت کد را می دهد به عنوان مثال یک موتور جستجو می تواند قسمتی از کد جستجو را بر روی یک ماشین دیگر انتقال دهد، ماشین دیگر حتما محتوی اطلاعات وب ها می باشد.

Receiver : به عنوان مثال اپلت های جاوا

E.a.t.p : در این روش کد بر روی ماشین مقصد فرستاده می شود، ماشین مقصد ماشینی است که نیاز به اطلاعات وب دارد.

E.i.S.P : در این روش بر خلاف روش قبلی که در آن کد بر روی ماشین درخواست کننده یا همان ماشین مقصد اجرا می شد ممکن است کد در یک ماشین مجزای دیگری اجرا شود.

نکته: در مهاجرت Weak فقط کد مهاجرت می شود.

نکته: ماشین دیگر وظایف ماشین مبدا را در Strong انجام میدهد.

اگر Sender شروع کننده باشد، Client یک query برای سرور می فرستد، Client باید ثبت شده باشد، Receiver می تواند نامعلوم باشد.

موجودیت مهاجرت شده چگونه اجرا می شود؟

یا بر روی پردازه مجرای جدید(target) اجرا می شود یا بر روی ماشینی که وجود داشت اجرا می شود.

شبیه سازی(Remot Cloning) : یک کپی از پردازه ایجاد می کنند و آن کپی اجرا می شود.

Migrate the Process : خود پردازه انتقال داده می شود.

اگر مهاجرت Strong باشد هم کد و هم اجرایی مهاجرت می کند.

آیا منابع هم مهاجرت می کنند؟

بستگی به طریق نگاشت پردازه به منابع دارد، یعنی رابطه بین پردازه و منابع چگونه باشد.

انواع وابستگی بین پردازه و منابع

- منابع به وسیله شناسه شان شناخته می شود، URL ، FTP Server (قوی)
- به وسیله مقدارشان، کتابخانه های جاوا
- به وسیله نوع، پرینتر و سرویس های محلی(ضعیف)

تقسیم بندی از جهات دیگر(وابستگی بین پردازه و منبع)

Unattached : منابع به هیچ نودی وابسته نیستند مثل Data ، یعنی منبع به پردازه وصل نیست.

Fastened : می تواند مهاجرت انجام دهد با هزینه زیاد، مثا Data base، Web Sites

Fixed resources : مثل پرینتر، وسایل محلی، پورت ها و ... منابع ثابتند

اعمال مهاجرت منابع:

- MV (انتقال منبع، move)
- GR (Global Resource): یعنی همه به آن دسترسی داشته باشند.
- RB (Rebind) : نگاشتی بین پردازه و منبع جدید به وجود بیاید
- CP : باید عمل کپی انجام شود.

نکته: اگر منبع Fix باشد انتقال منبع وجود ندارد.

مهاجرت در سیستم های ناهمگن:

سیستم های ناهمگن Platform متفاوت دارند، یعنی هم از لحاظ معماری و هم از لحظ OS تفاوت دارند. اگر سیستم ناهمگن باشد فقط Weak mobility را پشتیبانی می کند و Strong mobility خیلی سخت است

- Weak mobility هم کد دوباره کامپایل می شود و هم باید اطلاعات زمان اجرا نداشته باشد
- در Strong mobility قسمت Code Segment باید کامپایل شود و سگمنت اجرا(پشته) هم باشد ارسال شود.
- ماشین مجازی کد میانی تولید می کند مثل جاوا یا این که تفسیری باشد مثل اسکریپت ها

نکته: جاوا در هر ماشینی اجرا می شود چون کد میانی تولید می کند و وابسته به ماشین نیست

الگوریتم برکلی:

مناسب ماشین هایی که هیچیک زمان WWV دریافت نمی کنند و خود زمان ها را همگام می کنند

الگوریتم Lamport چه مشکلاتی دارد؟

1- ممکن است Timestamp ها یکسان باشند، پردازه هائی که مستقل هستند ممکن است ساعت یکسانی داشته باشند. Message ها بایستی بر اساس Timestamp مرتب شوند

راه حل آن این است که یک Timestamp منطقی سراسری یا واحد در نظر بگیریم، جهت مرتب سازی سراسری Timestamp ها نیاید یکسان باشند که دو قسمت دارد.

- Ti : Timestamp ی که ساعت Lamport ایجاد می کند.
- i : id پردازه که سراسری است و id هیچ دو پردازه ای نباید مثل هم باشد

جهت مقایسه:

(Ti,i)<(Tj,j)
If and only if
Ti<Tj or Ti=Tj and i < j

در حقیقت برای رفع این مشکل از Lamport تعمیم یافته(ساعت سراسری) استفاده می کنیم.

2- اگر event(A)<event(B) باشد نمی توانیم یک نتیجه سراسری از Lamport بگیریم مرتب کردن Timestamp های event های مستقل امکان پذیر نیست و نمی توانیم نتیجه بگیریم که A زودتر انجام شده است. چون ساعت سراسری نداریم. برای event های وابسته می توانیم نتیجه بگیریم.

راه حل. استفاده از بردار آرایه ساعت

مقدار دهی آرایه ساعت:

- اگر n سیستم داشته باشیم اولین بار مقدار ارایه را صفر قرار می دهیم، هر پردازه یا سیستمی بردرا ساعت مخصوص به خود را دارد.
- زمانی که یک event روی یک پردازه یک سیستم اجرا می شود به خانه مربوط به آن سیستم یک واحد اضافه می شود، هر خانه آرایه مربوط به یک سیستم است.
- وقتی یک message از یک سیستم به سیستم دیگر ارسال می شود بردار ساعت هم با آن ارسال می شود.
- فرض می کنیم message(Vi) به همراه بردار ساعتش ارسال شود، یعنی این message از سیستم Pi برای سیستم Pj ارسال شده باشد، سیستم Pj وقتی message را دریافت می کند بردار های ساعت آنها را مقایسه می کند که خواهیم داشت

$V_j[i] = \max(V_i[i], V_j[i])$ for $i = 1,2,...$ (بردار ساعت مربوط به پردازه Pj)

نکته1: دو بردار در صورتی مساوی هستند که تمام خانه هایشان باهم برابر باشند.

نکته2: $V \le V'$ اگر تمام خانه ای متناظر باهم کوچکتر مساوی باشند

نکته3: اگر e قبل از e' اتفاق افتاده باشدباید $V(e) < V(e')$ باشد

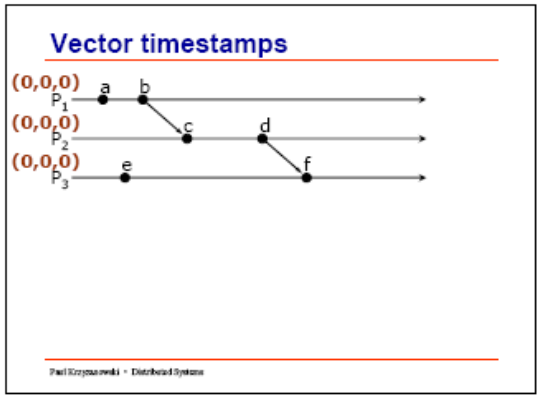
نکته4: مشابه نکته 4 در الگوریتم لم پورت: اگر $V(e) < V(e')$ بایستی e قبل از e' اتفاق افتاده باشد.

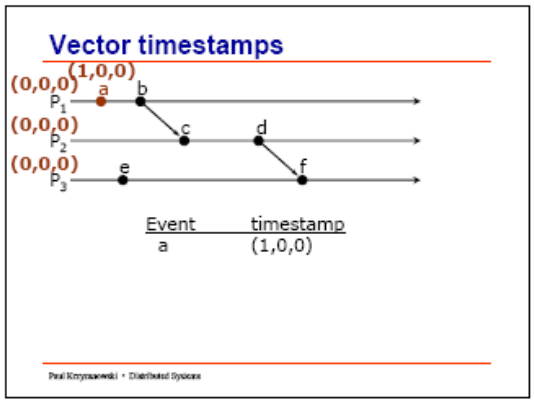
نکته5: به دو event همروند گوئیم اگر نه $V(e) < V(e')$ و نه $V(e') < V(e)$ باشد.

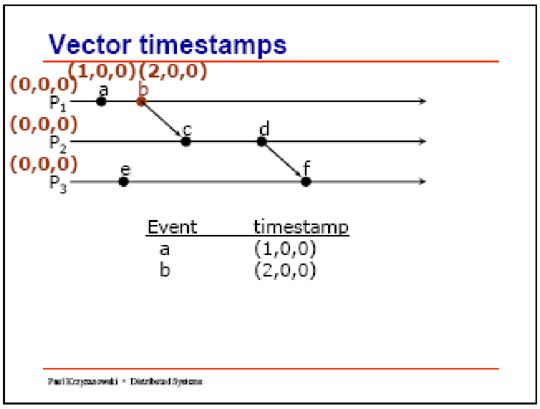
نکته6: تعداد پردازه ها= تعداد خانه های بردار هر پردازه

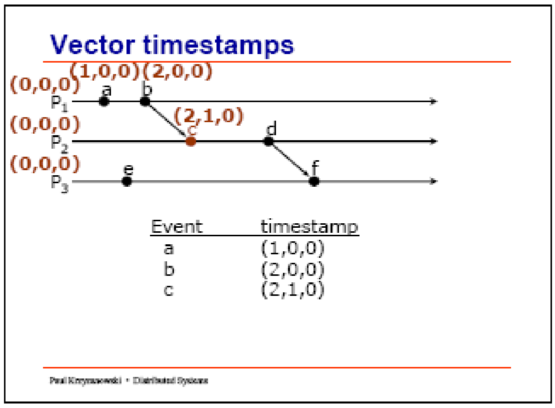
برای هر پردازه یک بردار داریم، یعنی برای سه پردازه سه بردار سه خانه ای داریم.

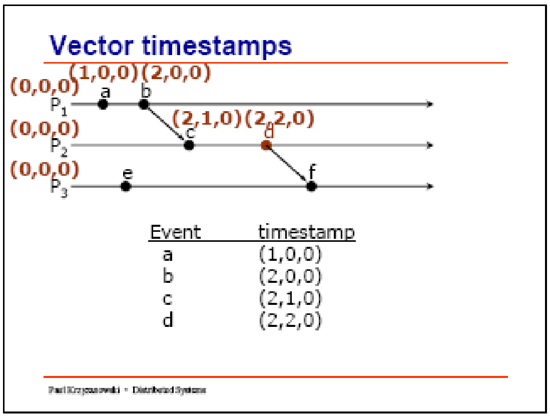
مثال. در زیر مرتب سازی event ها و event های همروند مشخص شده اند.

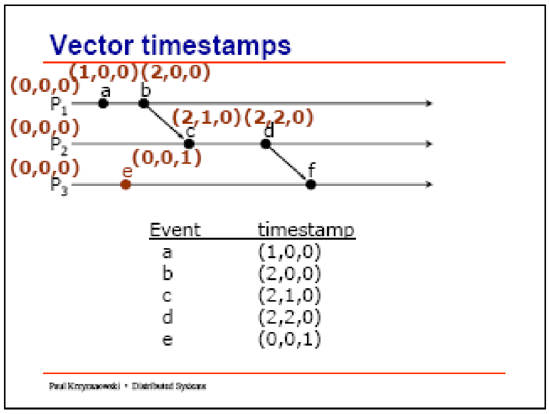


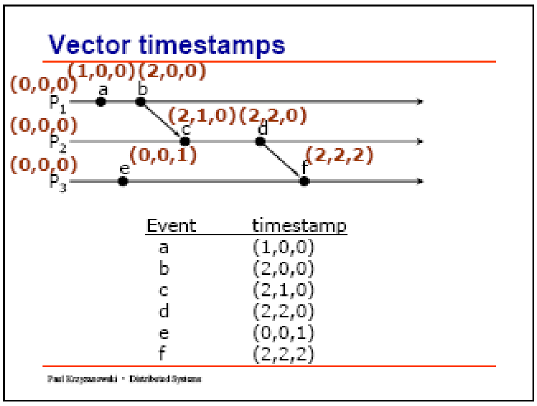


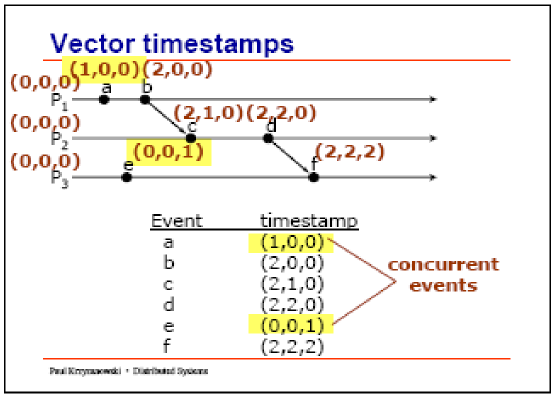


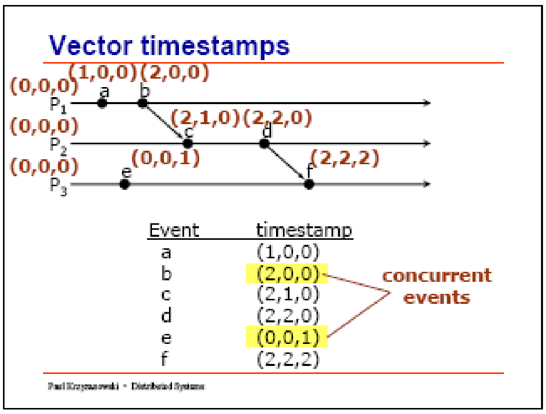


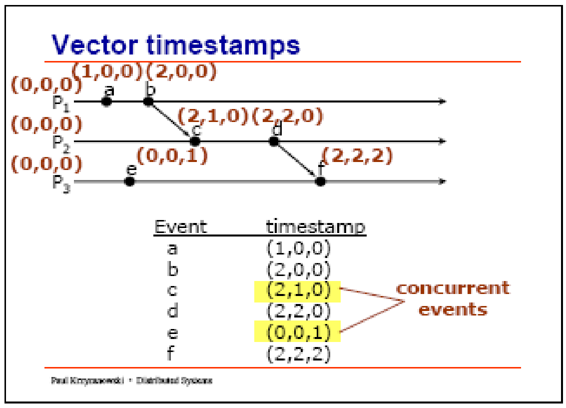












بن بست پردازه ها:

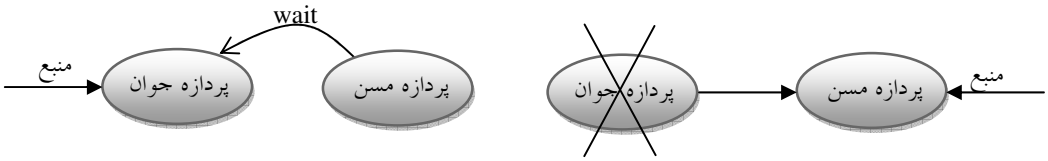
اگر بین پردازه ها بن بست رخ دهد دو حالت برای رفع بن بست وجود دارد.

- Wait-die
- Wound-wait

Wait-die : در این حالت یک پردازه جوان و یک پردازه مسن داریم آنکه IP کمتری داشته باشد یا Timestamp بیشتری داشته باشد یا زودتر درخواست دهد جوان است

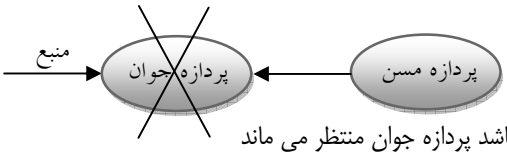
اگر بن بست رخ دهد دو حالت وجود دارد.

1- منبع در اختیار پردازه جوان است و پردازه مسن هم نیاز به منبع دارد بنابراین پردازه مسن منتظر می ماند.
2- منبع در اختیار پردازه مسن است و پردازه جوان هم نیاز به منبع دارد در این صورت پردازه جوان خودکشی می کند.



wound-wait :

- اگر منبع در اختیار پردازه جوان باشد و پردازه مسن هم نیاز به منبع داشته باشد پردازه مسن پردازه جوان را می کشد



- اگر منبع در اختیار پردازه مسن باشد و پردازه جوان هم نیاز به منبع داشته باشد پردازه جوان منتظر می ماند