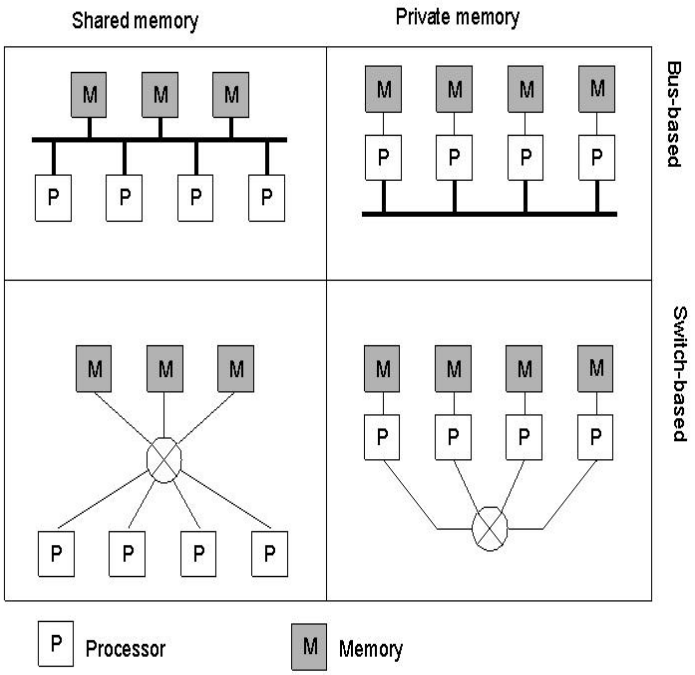
Distributed System

سیستم های توزیع شده به دو گروه اصلی تقسیم می شوند. سیستم‌های چند پردازنده‌ای یا Multi Processor و سیستم‌های چند کامپیوتری یا Multicomputer .

در این بخش به بررسی تفاوت‌های این دو سیستم می‌پردازیم.

MultiProcessor سیستمی است که درآن یک فضای مشترک فیزیکی حافظه برای تمامی processor ها وجود دارد. حال اینکه این فضا چگونه تقسیم شده باشد و دسترسی به آن از چه تکنیکی باشد جای بحث دارد. یکی از ساده ترین تکنیک های موجود، استفاده از یک Bus مشترک بین حافظه و تمامی پردازنده هاست که به عنوان نمونه در شکل نشان داده شده است. با توجه به شکل، تمامی پردازنده ها، تمامی ماژول‌های حافظه‌ی موجود را به یک شکل دیده و تمامی حافظه بین تمامی پردازنده ها Share شده است.



در Multicomputer هر پردازنده‌ای یا هر ماشینی دارای یک حافظه خصوصی (Privatememory) است. گرچه دسترسی به آن حافظه برای ماشین‌های دیگر ممکن خواهد بود اما این نحوه دسترسی به مانند دسترسی انجام شده در Multiprocessor خواهد بود. در واقع در شکل این مسأله نشان داده شده است که هر پردازنده یک حافظه خصوصی دارد.

سیستم‌های Multicomputer و همچنین Multiprocessor لزوماً با تکنیک های مبتنی بر Bus طراحی و مدل نمی‌شوند. استفاده از Switch می‌تواند تکنیک دیگری برای این قضیه باشد. در مورد Multicomputer و Multiprocessor استفاده از Switch نیز امکان پذیر است. وقتی که از Switch استفاده می شود، دسترسی به حافظه ها در سیستم Multi processor از طریق Switch انجام می شود و در سیستم Multicomputer ارتباط بین Computer ها از طریق Switch خواهد بود. نکته ی دیگر این است که سیستم‌های Multicomputer را به دو گروه عمده تقسیم می‌کنیم: ۱- سیستم های همگن یا Homogeneous و ۲- سیستم های غیرهمگن یا Heterogeneous. .

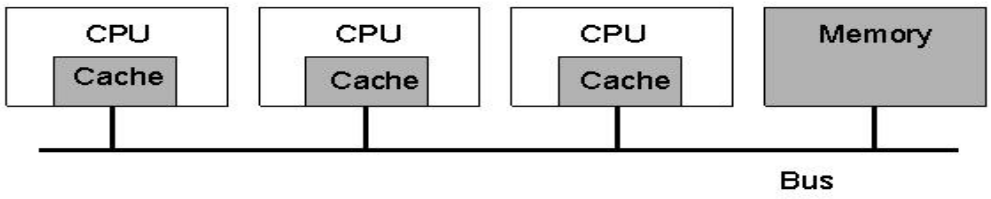
در یک سیستم همگن (مانند سیستم های موازی) ، تکنولوژی یکسانی در سر تا سر سیستم توزیع شده وجود دارد. به عبارت ساده‌تر تمامی ماشین ها یا Computer هایی که در سیستم توزیع شده Multicomputer ، حضور دارند همگی از یک جنس می باشند و همگی سیستم های عامل مشابهی را استفاده می کنند.

در سیستم های غیرهمگن، کامپیوترهایی که از سخت افزارهای متفاوتی تشکیل شده اند و احیاناً سیستم های عامل متفاوتی نیز بر روی هر تک کامپیوتر در حال اجراست شبکه را تشکیل داده اند. باید دقت نمود زمانیکه بیان می شود، کامپیوترهای متفاوت در سیستم غیر همگن، هدف ما این نیست که در واقع سطح توانایی سخت افزاری کامپیوترها متفاوت است. یعنی به اشتباه فرض کنید (به عنوان مثال) کامپیوتری از نوع pentium و دیگری از نوع ٤٨٦ باعث ایجاد یک سیستم غیرهمگن می شود. منظور از کامپیوترهای مختلف، کامپیوترهائی است که اصولاً معماری آنها با هم یکسان و احیاناً سازگار نیست. به عنوان مثال، فرض کنید کامپیوتری در یک

سیستم همگن یک کامپیوتر Main frame و سیستم دیگری یک کامپیوتر PC باشد، این دو از نظر سخت افزاری با یکدیگر سازگاری ندارند.

Multiprocessor

در یک سیستم Multiprocessor یا چند پردازنده که با هر یک از معماری های مطرح شده ممکن است ایجاد شود، یکی از مشکلات اساسی که در معماری مبتنی بر مدل Bus وحتی مبتنی بر مدل Switch ممکن است پیش آید این است که Bus و Switch مربوطه احتمالاً در شرایطی ممکن است دچار overload یا سر بار بیش از حد کاری شود. این مسأله راه حل هایی را به دنبال دارد. از جمله راه حل هایی که در شکل نشان داده شده است، استفاده از یک local Cache یا یک حافظه نهان محلی با سرعت بالا به ازای هر پردازنده است.

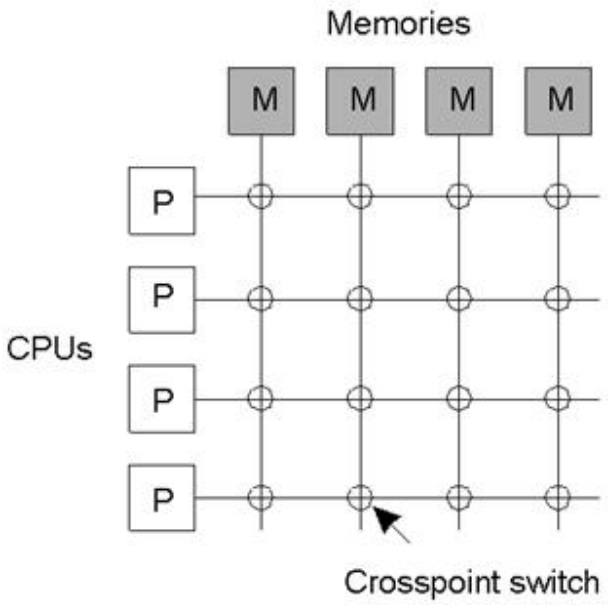


بدیهی است که این Cache ها قسمتی از حافظه ی اصلی را در خود نگهداری می کنند و چه بسا که محتوای تمامی Cache ها در شرایطی نیز یکسان باشد. مشکلی که در تمام سیستم های مبتنی بر Cache وجود دارد بحث سازگاری و همگام بودن Cacheها و cache coherency است. با استفاده از یکی از تکنیک های cache coherency باید cache های مربوط به CPU ها با یکدیگر همسان و همگام شوند.

یکی از مشکلات سیستم های مبتنی بر cache و سیستم هایی که تکنیک Caching را به کار می برند بحث limited Scalability یا عدم امکان گسترش سیستم می باشد. در واقع گسترش سیستم در اینجا وابسته به

تعداد cacheها و نحوه ی مدیریت cache coherency خواهد بود. برای رفع مشکل Scalability از تکنیک های دیگری نیز می توان استفاده کرد. می توان مفهوم Bus را به طور کلی حذف نمود و سیستم را به سمت Switch حرکت داد.

Switch ها انواع مختلفی دارند که به طور کلی دو نوع Switch مورد نظر قرار می گیرد. یکی از این Switch ها، Switchهای Crossbar است که در شکل های قبلی نمونه ای از آن را دیدیم. در Switch Crossbar کماکان مشکل Bus ممکن است وجود داشته باشد و حجم زیادی از Load در شرایطی بر روی این Switch قرار گیرد. نوع دیگر Switch ها Cross point switch است. این Switch یا سوئیچ های نقطه ای، سوئیچ هایی هستند که در واقع اتصال نقطه‌ای بین دو ند را برقرار می کنند. در شکل نمونه ای از استفاده از سوئیچ های Cross point را مشاهده می کنید.

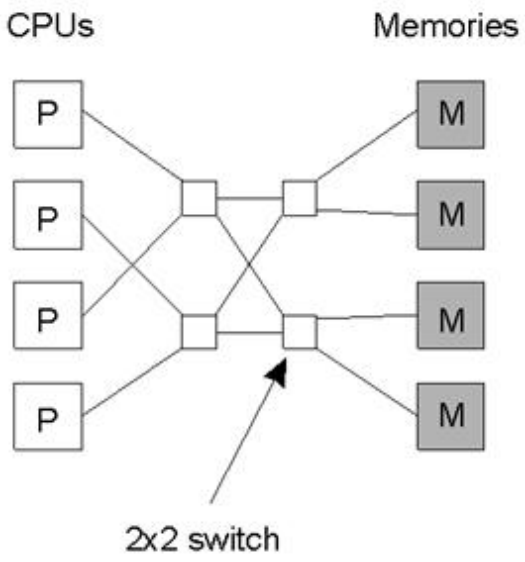


(a)

همانطور که در شکل می بینید Switch های Cross point نقاط اتصال در یک Grid هستند که به نوعی چهار طرف خود را به یکدیگر متصل می‌کنند. بنابراین چنانچه بخواهیم از مفهوم Switch های Cross point استفاده کنیم، آنگاه نیازمند Grid بندی پردازنده ها و حافظه های موجود خواهیم بود و همانطور که گفته شد در محل اتصال هر Grid یک سوئیچ نیاز است. بنابراین یکی از مشکلات استفاده از سوئیچ های Cross point تعداد زیاد سوئیچ هاست، به صورتی که به اندازه ی n^2 سوئیچ مورد نیاز خواهد بود. در اینجا نیز راه حلی می توان ارائه داد و آن استفاده از مفهوم Omega connection است. Omega connection مفهومی است که برای رفع مشکل تعداد سوئیچ های Cross point در یک سیستم توزیع شده مورد استفاده قرار می‌گیرد.

Omega connection

Omega connection در واقع یک سوئیچ، Multi stage است، یا به عبارت دقیق تر معماری مبتنی بر Omega connection یک معماری Multi stage یا چند مرحله ای است. بنابراین چنانچه قرار باشد با استفاده از ارتباطات Omega یک پردازنده از داده های مربوط به یک حافظه بهره گیرد، آنگاه دسترسی به داده ها احتمالاً با چندین مرحله عبور از سوئیچ های مختلف صورت خواهد گرفت؛ بنابراین برای اینکه این تعداد مراحل پشت سر هم سیستم را دچار مشکل نکند و سرعت سیستم را تحت تأثیر قرار ندهند، سوئیچ ها باید سریع باشند. در شکل نحوه ی ارتباط Omega connection نشان داده شده است. با توجه به شکل، علت نامگذاری این سوئیچ‌ها به نام Omega شکلی است که برای ارتباطات آنها در نظر گرفته شده است و به نوعی می توان آن را مشابه نماد Omega دانست. قطعاً این مفهوم و نحوه ارتباط سوئیچ‌ها در Omega connection خاطره ی مربوط به Rotterها در یک شبکه را زنده خواهد کرد.



(b)

نکته مورد بحث در Omega connection ، هزینه‌ی سوئیچینگ است و زمانیکه برای سوئیچینگ مصرف می‌شود، و با افزایش تعداد پروسسورها و ماژول‌های حافظه این زمان افزایش خواهد یافت. یکی از راه حل‌هایی که برای رفع این مشکل ارائه می شود استفاده از مفهوم NUMA (Non Uniform Memory Access) یا دسترسی غیریکنواخت به حافظه است. در NUMA به ازای هر پروسسور باید یک حافظه محلی نیز در نظر گرفت. با توجه به مباحث قبل دریافتیم که در مفهوم NUMA ، از تکنیک Caching برای حل مشکل BUS overload استفاده کردیم. در اینجا نیز تکنیک، مشابه است با این تفاوت که حافظه‌ای که برای هر پردازنده در نظر می‌گیریم یک حافظه ی محلی است و به معنای Cache نیست. تفاوت Cache با یک حافظه‌ی محلی در این است که کپی یک حافظه بزرگتر است. اما این حافظه محلی لزوماً کپی حافظه ی دیگری نیست و با در نظر گرفتن اینکه هر پردازنده یک حافظه محلی دارد و آنگاه پردازنده ها عملیات مربوط به خود را می توانند با دسترسی های سریع محلی انجام دهند لذا دسترسی به داده های خاص و عملیات بینابینی و داده های موقتی، دیگر نیازی به عبور از سوئیچ های مربوط به Omega نخواهد داشت. دسترسی به سایر حافظه ها یعنی حافظه های مشترکی که به عنوان ماژول های جداگانه پشت سوئیچ قرار دارند با کندی صورت خواهد

گرفت. اما ترکیب این دو مفهوم با یکدیگر باعث خواهد شد که میانگین زمان دسترسی به حافظه کاهش یابد و بنابراین این تکنیک، تکنیک کارا و مناسبی است.

مفاهیم نرم افزاری سیستم های توزیع شده

از دید کلان، نرم افزارهای سیستم توزیع شده دو وظیفه اصلی را بر عهده دارند:

۱- مدیریت منابع یا Resource Management

۲- سعی بر پنهان سازی غیرهمگن بودن سخت افزارهای موجود در لایه ی پائینی یک سیستم توزیع شده است.

در مطالب پیشین، خواصی مانند Openness و Transparent بودن در مورد سیستم های توزیع شده عنوان شد که قطعاً از جمله مواردی است که در طراحی نرم افزارهای مربوطه مورد توجه قرار می گیرند. اما سیستم‌های عاملی که برای سیستم های توزیع شده یا کامپیوترهای توزیع شده مورد توجه قرار می‌گیرند را به دو گروه عمده می توان تقسیم کرد:

۱- سیستم‌های Tightly-coupled یا سیستم های جفت محکم(اتصال قوی) ۲- سیستم های Loosely-coupled یا سیستم های جفت آرام یا روان(اتصال ضعیف).

سیستم های Tightly-coupled سعی می کنند که یک دید منفرد و عمومی از همه منابع آماده کنند و معمولاً این سیستم‌ها را Dos (Distributed Operating system) می‌نامند. کاربرد سیستم‌های DOS در Multiprocessor و Multicomputer های همگن است. اما سیستم های عامل Loosely-coupled سعی می کنند که مجموعه‌ای از کامپیوترها را مدیریت کنند که هر یک از این کامپیوترها سیستم عامل محلی خود را دارند. بنابراین سیستم های عامل توزیع شده در اینجا وظیفه خواهند داشت که سعی کنند این سیستم‌های عامل را به نوعی همگام و هماهنگ کنند. به عبارت دیگر امکانی فراهم شود که

این سیستم های عامل سرویس ها و منابع خود را برای دیگران ارائه کنند و در واقع امکان فراهم بودن آنها را برای دیگران مهیا سازند. لذا در این نوع سیستم‌ها اصولاً سیستم توزیع شده باید در سیستم‌های عامل محلی Embed شوند. این نوع سیستم های عامل را معمولاً NOS یا Network Operating System می نامند، بنابراین سیستم های عامل شبکه در این گروه قرار می گیرند.

Middle ware یا میان افزار

مفهوم نرم افزاری دیگری که در سیستم های توزیع شده وجود دارد مفهومMiddleware یا میان افزار است. هدف یک Middleware ارائه شفافیت بهتر است . به عبارت دیگر، هدف Middleware ارائه لایه ای است که این لایه سطح شفافیت سیستم را افزایش می دهد و معمولاً این سطح شفافیت در مباحث مربوط به سرویس هایی است که از قسمت های مختلف سیستم در اختیار قسمت‌های دیگری قرار می گیرد. حال می‌توان این سه مفهوم نرم افزاری را با یکدیگر در قالب یک جدول مقایسه کرده و خصوصیتهای آنها را مقایسه نماییم.

هدف اصلی	توصیف	سیستم
مخفی کردن و مدیریت منابع سخت افزاری	یک سیستم عامل اتصال قوی برای چند پردازنده‌ها و چند کامپیوترهای همگن	سیستم عامل توزیع شده
ارائه خدمات محلی به سرویس گیرنده های راه دور	یک سیستم عامل اتصال ضعیف برای چند کامپیوترهای نا همگن(LAN و WAN)	سیستم عامل شبکه‌ای
ایجاد شفافیت توزیع	یک لایه اضافه شدنی بر بالای سیستم عامل شبکه‌ای که سرویس‌های عمومی را ارائه می‌کند.	میان افزار

در مباحث قبل عنوان شد که سیستم DOS یک سیستم عامل Tightly-coupled برای Multiprocessor ها و Multicomputer های همگن است که هدف اصلی آن پنهان سازی و مدیریت منابع سخت افزاری است. سیستم NOS، Loosely-coupled می‌باشد که برای سیستم‌های غیرهمگن و Multicomputerهای غیرهمگن مانند شبکه‌های LAN & WAN استفاده می‌شود که در واقع هدف اصلی آن ارائه سرویس های محلی به مشتری های راه دور است. با توجه به مطالب پیشین قرار شد:

Middleware لایه‌ی اضافه یا جدیدی باشد که معمولاً بر روی NOS قرار می‌گیرد و معمولاً شامل سرویس‌های عمومی می باشد که احتمالاً تمامی افراد موجود در سیستم نیاز به استفاده از آنها دارند. هدف اصلی یک Middleware آماده کردن شفافیت توزیع و یا افزایش سطح شفافیت سیستم است.

حال به بررسی جزئیات این مباحث می‌پردازیم:

DOS یا Distributed Operating Systam هم برای Multiprocessorها و هم برای Multicomputer ها قابل استفاده است. کارایی DOS شبیه سیستم‌های عامل سنتی است و هدف آن مدیریت چندین CPU است. در Multiprocessor operating system از نوع DOS می توان گفت، که این سیستم عامل یک توسعه بر روی سیستم های عامل تک پردازنده است. در چنین سیستمی همه‌ی ساختارهای داده‌ای که برای مدیریت سخت افزار و پردازنده‌ها مورد استفاده قرار می‌گیرند در حافظه اشتراکی(Share memory) قرار می‌گیرند. بدیهی است که داده‌هایی که در Share memory قرار می‌گیرد باید در مقابل دسترسی‌های هم روند محافظت شوند، بنابراین تکنیک‌های مربوط به محافظت در اینجا قابل استفاده خواهد بود.

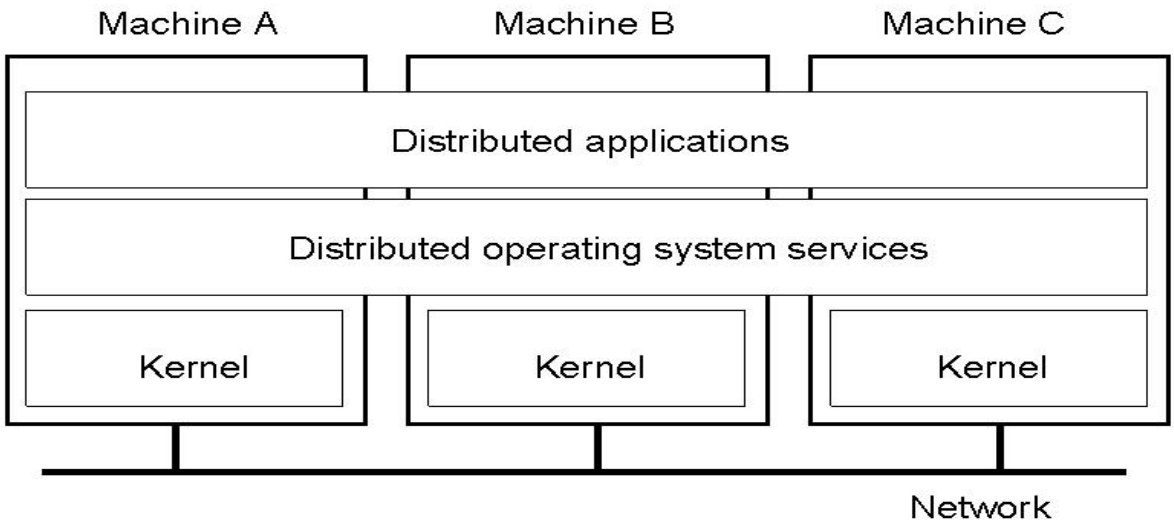
DOS

هدف اصلی در یک سیستم عامل DOS از نوع Multiprocessor افزایش Performance است و از دیگر اهداف مهم Transparent کردن تعداد CPU هاست . بطوری که کاربر نتواند بفهمد که چندین CPU در

لایه ی پایین در حال کار با یکدیگر هستند. در این گونه از سیستم ها تمامی ارتباطات با مدیریت داده‌ها در حافظه‌ی اشتراکی انجام می‌شود و همانطور که عنوان شد، نیازمند محافظت داده‌ها در مقابل دسترسی‌های همروند برای جلوگیری از ایجاد Raise condition و ایجاد مفاهیم دو به دو ناسازگاری است. استفاده از دو دو مفهوم پایه‌ای مانند Semaphore و Monitor در این قسمت می‌تواند کار را تسهیل نماید.

DOS در Multicomputer ها:

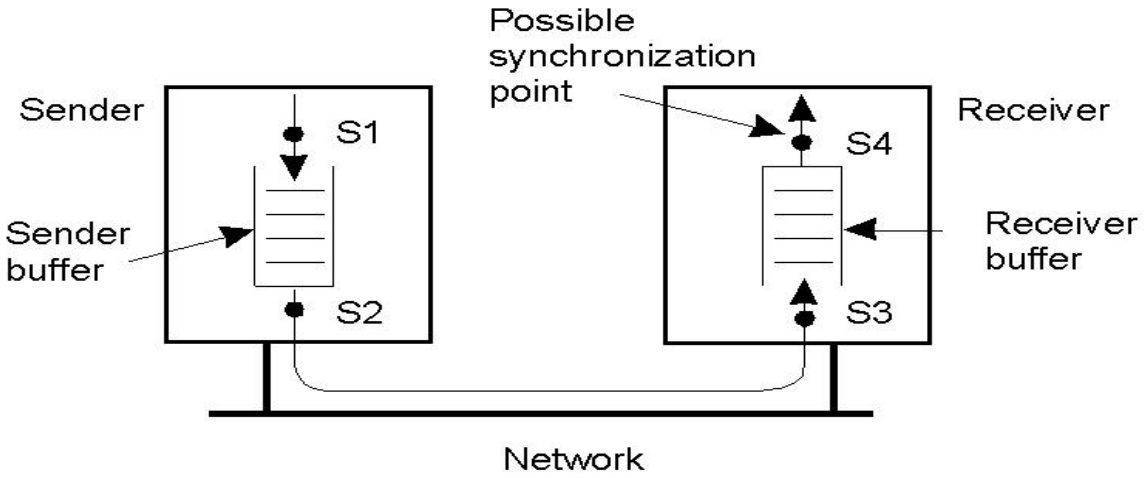
در چنین سیستم هایی، ساختمان‌های داده‌ای نمی‌توانند فقط در حافظه‌ی فیزیکی Share شوند، به نوعی ساختمان‌های داده‌ای در اینجا بر روی ماشین های مختلف قرار دارند و تنها وسیله‌ی ارتباطی بین ماشین‌ها استفاده از تکنیک‌های ارسال پیغام یا Message Passing است. نمونه‌ای از یک سیستم DOS و نحوه‌ی چیدمان آن بر روی ماشین های مختلف در شکل نشان داده شده است.



با توجه به شکل، ۳ ماشین A,B,C وجود دارد که هر یک به طور مستقل می توانند عمل کنند و سیستم عامل توزیع شده، در داخل یک لایه بر روی هر سه این ماشین‌ها قرار دارد و در یک لایه‌ی بالاتر Applicationهای توزیع شده قرار دارند. بدیهی است که بر روی هر ماشینی امکان اجرای برنامه های محلی

غیر توزیع شده نیز وجود دارد. اما همانطور که عنوان شد، ارتباط بین ماشین ها در یک سیستم توزیع شده نیازمند استفاده از مفاهیم Message Passing است. همگام سازی پیغام ها و ماشین های مختلف در راستای مفاهیم Message Passing دارای نکاتی است که در اینجا بدان اشاره می شود

. با توجه به شکل، هنگامی که یک ماشین پیغامی را برای ماشین دیگر از طریق شبکه ارسال می کند و یک ماشین Sender و یک ماشین Receiver وجود دارد، استفاده از Buffer در هر دو سمت Sender و Receiver کار متداولی است. حال Sender معمولاً پیغام را در داخل Buffer send قرار می دهد و Receiver پیغام را از بافر Receive یا بافر دریافت می تواند دریافت کند.



آنچه که در این جا حائز اهمیت است این است که با توجه به این شکل Sender ممکن است در نقاط S1,S2,S3 و حتی S4، block شود. به این معنی که Sender پس از فرستادن پیغام در صورتی می تواند به کار خود ادامه دهد که به یکی از این نقاط رسیده باشد. به عبارت دیگر پیغام ارسال شده در یکی از این نقاط قرار گرفته باشد. اما Receiver یا دریافت کننده تنها در نقطه ی S3، block می شود.

به توضیح بیشتر این مفهوم می پردازیم:

نقطه ی S1 نقطه ای است که نقطه ی آغازین بافر Send است. اگر می گوئیم Sender در نقطه ی S1، block می شود به این معنی است که Sender پیغام را ارسال می کند و هنگامی عملیات خود را ادامه می‌دهد که پیغام مربوطه بتواند در Buffer send قرار گیرد و اگر پیغام مربوطه در بافر Send قرار گرفت Sender به کار خود ادامه می‌دهد. در واقع اگر Buffer send به گونه ای بود که امکان دریافت پیغام را نداشت Sender، block می شود تا این اتفاق بیفتد. نقطهS2 یک نقطه‌ی خروج از Buffer send است به این معنی که اگر پیغامی که در بافر قرار گرفته است بوسیله سیستم عامل از بافر خارج شود و به سمت کلاینت ارسال شود ارسال کننده پیغام می تواند به کار خود ادامه دهد. این که یک پیغام چه مدت در بافر باقی می ماند وابسته به ملاحظاتی است که سیستم ها در رابطه با مدیریت بافر و مدیریت پیغام در نظر می گیرند. از جمله این ملاحظات می تواند مباحث مربوط به ترافیک شبکه و یا هر موضوع دیگری باشد.

نقطه S3 یک نقطه ی ورودی به بافر دریافت کننده یا Receiver است. در واقع اگر عنوان می‌شود که Sender در نقطه ی S3، block می شود به این معنی است که ارسال کننده پیغام عملیات خود را متوقف می‌کند تا وقتی که پیغام مورد نظر وارد بافر دریافت کننده شود. قطعاً اینکه چطور ارسال کننده ی پیغام متوجه این مسأله می شود که پیغام مورد نظر به بافر دریافت کننده رسیده است، ساز وکارهایی است که باید سیستم مدیریت ارسال پیغام در نظر بگیرد.

نقطه S4 به معنای برداشتن پیغام توسط دریافت کننده است و لذا اگر عنوان می‌شود که ارسال کننده در نقطه ی S4 متوقف می شود به این معنی است که اگر ارسال کننده پیغامی ارسال کند، هنگامی می تواند عملیات مربوط به خود را دنبال کند که پیغام مربوطه حتماً توسط دریافت کننده برداشته شود و مطمئن باشیم که این پیغام به مقصد رسیده است. در واقع اینکه کدام یک از این نقاط را تنظیم کنیم و به عنوان نقطه‌ی ادامه حرکت Sender در نظر بگیریم جزء مواردی است که سطح ارتباط Reliable سیستم را افزایش می‌دهد. زمانی که بیان می‌شود Receiver در نقطه S3 می تواند Stop کند به این معنا است که اگر دریافت کننده‌ای به قصد دریافت یک

پیغام وارد فضای دریافت شود در نقطه S3 متوقف می شود، به این معنی که به محض اینکه پیغامی وارد بافر مربوطه شود دریافت کننده می تواند پیغام مربوطه را برداشته و عملیات مربوط به خود را دنبال کند.

تأکید می شود که معمولاً در سیستم ها امکان تنظیم هر یک از این نقاط وجود دارد. به این معنی که تنظیم کنیم، Sender به طور خاص بر روی کدام یک از این نقاط متوقف شده و تعاریف مربوطه را در ارتباط Reliable با دریافت کننده داشته باشد.

مفاهیم عنوان شده را می‌توان به طور خلاصه در یک جدول بیان کرد. نقطه‌ی همزمانی (Synchronization) همان S1,S2,S3,S4 است. اولین نقطه، نقطه S1 است که Sender تا وقتی که بافر Full نیست، block می‌شود. در این مرحله Send buffer مورد استفاده قرار می گیرد و این نقطه در واقع یک ارتباط Reliable را لزوماً گارانتی نمی‌کند. نقطه دوم یا S2 نقطه‌ای بود که Sender تا وقتی که message ارسال شود در واقع مورد block قرار می‌گیرد. در اینجا دیگر مفهوم Send buffer مورد توجه نیست و بافر ارسال بررسی نخواهد شد. ولی باز هم ارتباط در واقع Reliable لزوماً برقرار نیست و گارانتی نخواهد شد. اما اگر در نقطه‌ی S3 که Sender متوقف می شد تا وقتی که پیغام مورد نظر به آن نقطه برسد به عنوان نقطه Synchronization انتخاب گردد آنگاه می‌توان گفت یک ارتباط Reliable به طور ۱۰۰٪ ایجاد می‌شود. همین قضیه برای هنگامی که Sender پیغام مورد نظرش به دست دریافت کننده برسد (deliver)، اگر این نقطه به عنوان نقطه ی همگام سازی انتخاب شود باز هم در واقع یک communication یا یک ارتباط Reliable را گارانتی کرده ایم.

تضمین ارتباط قابل اعتماد	بافر ارسال	نقطه همزمانی S1,S2,S3,S4
غیر ضروری	بلی	مسدود کردن فرستنده تا زمان پر بودن بافر(S1)
غیر ضروری	خیر	مسدود کردن فرستنده تا زمان ارسال پیام(S2)
ضروری	خیر	مسدود کردن فرستنده تا زمان دریافت پیام(S3)
ضروری	خیر	مسدود کردن فرستنده تا زمان تحویل پیام(S4)