نظریه CAP:

در مباحث نظری علوم کامپیوتر، نظریه‌ی CAP (که نام دیگر آن نظریه‌ی Brewer هست)، بیان می‌کند که هر پایگاه‌داده‌ی (انبار داده) توزیع‌شده[1]، در بهترین حالت تنها می‌تواند فراهم کردن دو مورد از سه موارد زیر را تضمین کند.

۱. سازگاری (Consistency)

۲. دسترسی‌پذیری (Availability)

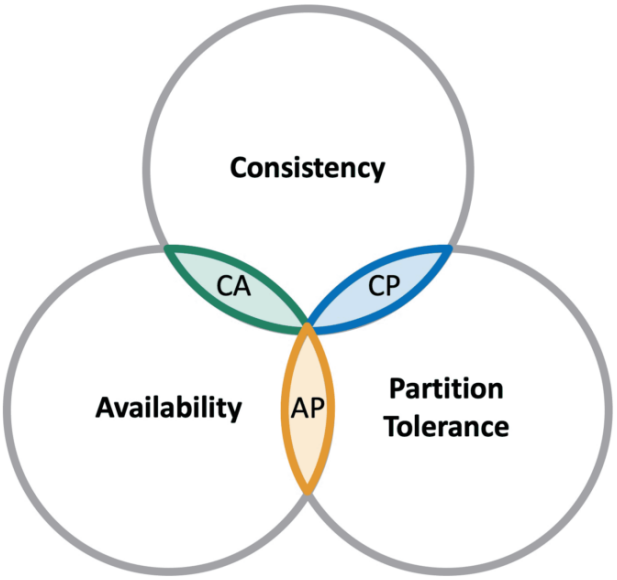
۳. تاب‌آوری تقسیم‌شدن (Partition Tolerance)

به بیان دیگر، یک سیستم توزیع‌شده نمی‌تواند در هر سه ویژگی سازگاری، دسترسی‌پذیری و تاب‌آوری تقسیم‌شدن) را داشته باشد. پس حالت‌های موجود یکی از سه حالت زیر است:

الف) سازگار و دسترسی‌پذیر باشند. (CA)

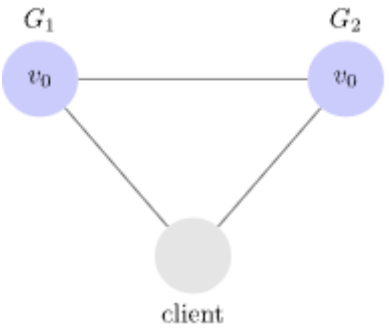
ب) سازگار و تاب‌آور بعد از تقسیم‌پذیری باشند. (CP)

پ) دسترسی‌پذیر و با تاب‌آوری بعد از تقسیم‌پذیری باشند. (AP)

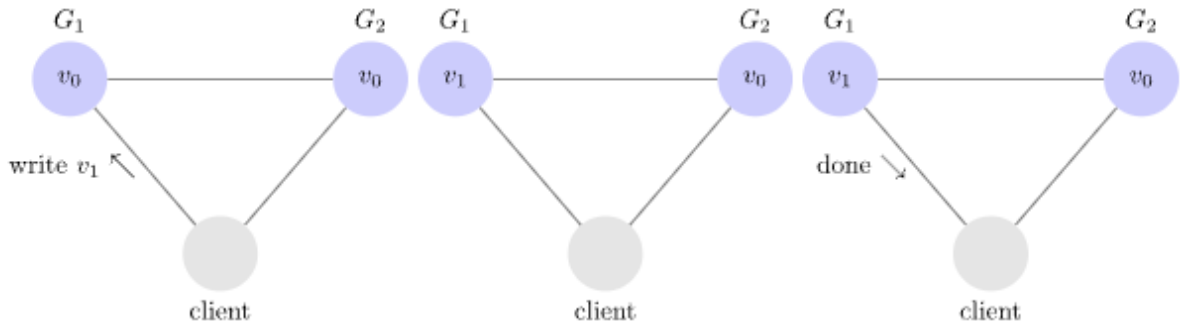


[1] Distributed data store

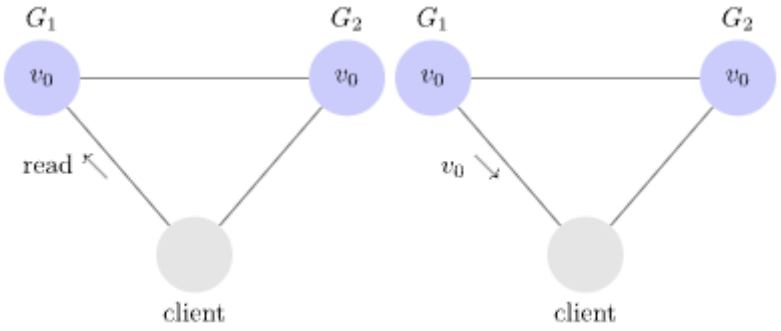
در مثال‌های قسمت‌های بعدی، فرض کنید یک سیستم‌توزیع شده متشکل از یک کاربر و دو گره‌ی G1 و G2 داریم که مقدار متغیر v را در خود نگه می‌دارند. مقدار اولیه‌ی v را فرض می‌کنیم v0 است.



کاربر می‌تواند درخواستی برای خواندن و نوشتن داده به هر کدام از گره‌ها ارسال کند. هنگامی که گره‌ای درخواستی را دریافت می‌کند، محاسباتی را که می‌خواهد انجام داده و سپس جوابی را برای کاربر ارسال می‌کند. برای مثال خواندن به این صورت انجام می‌شود:



خواندن نیز به شکل زیر انجام می‌شود:

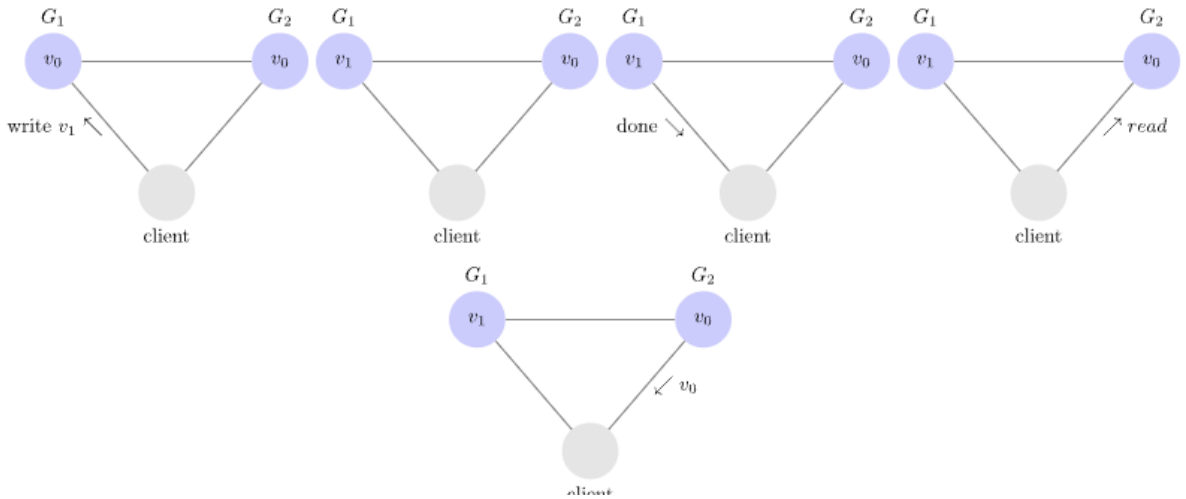


حال به تعریف هر کدام از موارد بالا به صورت مفصل می‌پردازیم.

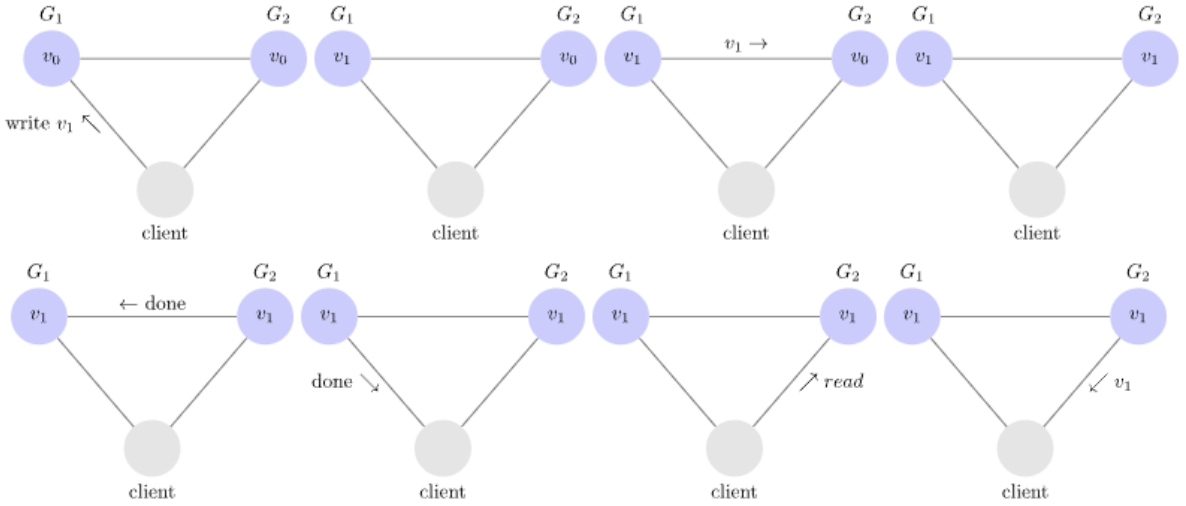
تعریف سازگاری:

بنا بر تعریف، نمای داده‌ها برای تمام کاربرهای سیستم توزیع‌شده باید به‌روز باشد (همه داده‌ها را یکسان ببینند.)؛ به بیان فنی‌تر، اگر درخواستی برای خواندن (read) داده ارسال شود، باید آخرین نوشته (write) و یا خطا را دریافت کنند. برای مثال، سیستم زیر ناسازگار است، زیرا در هنگام نوشتن داده، داده روی گره G1

ذخیره شده اما خواندن داده از روی گره‌ی G2 صورت گرفته که شامل آخرین نسخه‌ی داده نیست (به جای v1، v0 را دریافت نموده است.)



اما سیستم زیر سازگار است، زیرا داده بعد از نوشته شدن روی گره‌ی G1 به گره‌ی G2 ارسال شده و G2 نیز آخرین نسخه‌ی مقدار v یعنی v1 را دارد و دیگر فرقی ندارد که کاربر از روی کدام گره مقدار v را دریافت کند.

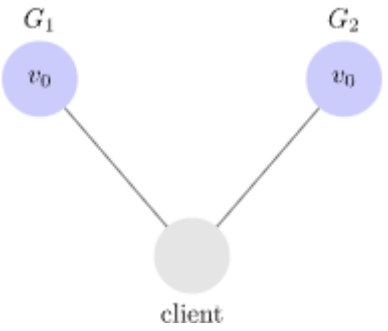


تعریف دسترسی‌پذیری:

بنا بر تعریف، داده‌ها همواره باید برای به‌روزرسانی و خوانده شده در دسترس باشند؛ به بیان فنی‌تر، باید همه‌ی خواندن‌ها شامل داده باشند، حتی اگر داده‌های دریافت‌شده جدیدترین نسخه از داده‌ها نباشد. برای مثال، اگر کاربر به گره G1 درخواستی ارسال کند و گره G1 خراب نباشد، گره G1 حتما باید جواب درخواست کاربر را بدهد و نباید درخواست کاربر را نادیده بگیرد.

تعریف تقسیم‌شدن:

بنا بر تعریف، سیستم توزیع‌شده باید بتواند در صورت خرابی شبکه که در آن همه‌ی اعضا نمی‌توانند به دیگر اعضا دسترسی پیدا کنند، به کار خود ادامه دهد؛ به بیان فنی‌تر، اگر ارتباط (connection) بین دو گره (node) از شبکه خراب شود (قطع شدن یا تاخیر در ارتباط)، خوشه (cluster، مجموعه‌ای از گره‌ها که با یک‌دیگر در سیستم توزیع‌شده کار می‌کنند) باید علی‌رغم هر تعداد خرابی در ارتباط، ادامه به کار کردن کند. برای مثال، اگر هر پیامی بین دو گره‌ی G1 و G2 حذف (drop) شود، حتی اگر همه‌ی پیام‌ها حذف شوند و سیستم به شکل زیر در بیاید، سیستم باید به‌صورت درست کار کند.

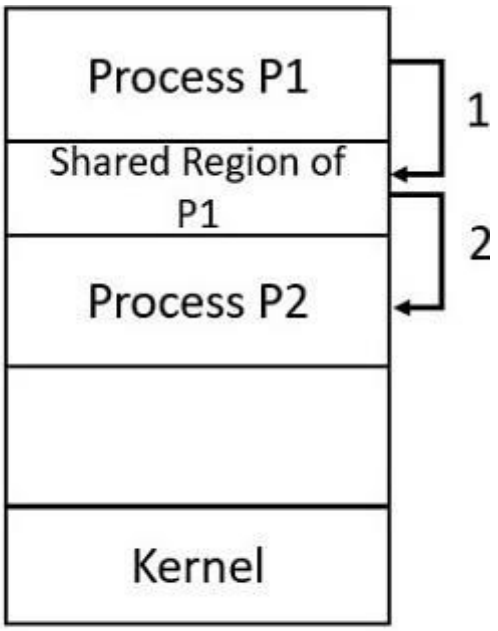


تفاوت بین سیستم‌های توزیع‌شده مبتنی بر ارسال متن و حافظه‌ی مشترک:

دو مدل اصلی ارتباط بین پردازه‌ها[2] در سیستم‌های توزیع‌شده که توجه‌های زیادی را به خود جلب کرده‌اند، مدل‌های مبتنی بر ارسال متن و حافظه‌ی مشترک هستند.

مدل حافظه‌ی مشترک:

در مدل حافظه‌ی مشترک، n پردازه با نوشتن و خواندن روی ثبات‌های مشترک، ارتباط برقرار می‌کنند. به بیان دیگر، پردازه‌ها با یک‌دیگر بر اساس ساخت یک فضای حافظه‌ی مشترک و به اشتراک گذاشتن آن ارتباط برقرار می‌کنند. اگر پردازه‌ای بخواهد ارتباطی را شروع کند و داده‌ای برای به اشتراک گذاشتن داشته باشد، آن‌گاه منطقه‌ی حافظه‌ی مشترک را در فضای آدرس آن ایجاد می‌کند. سپس اگر فرآیند دیگری بخواهد ارتباط برقرار کند و داده‌های اشتراکی را بخواند، باید خود را به فضای آدرس مشترک فرآیند آغازگر متصل کند.



Shared Memory System

مدل ارسال متن:

مدل ارسال متن سازوکاری را مهیا می‌کند که از طریق آن به پردازه‌ها اجازه می‌دهد تا با یک‌دیگر ارتباط برقرار کنند و کار[3]های خود را بدون نیاز به حافظه‌ی مشترک هم‌گام[4] سازند. در این مدل دو نوع عملیات داریم:

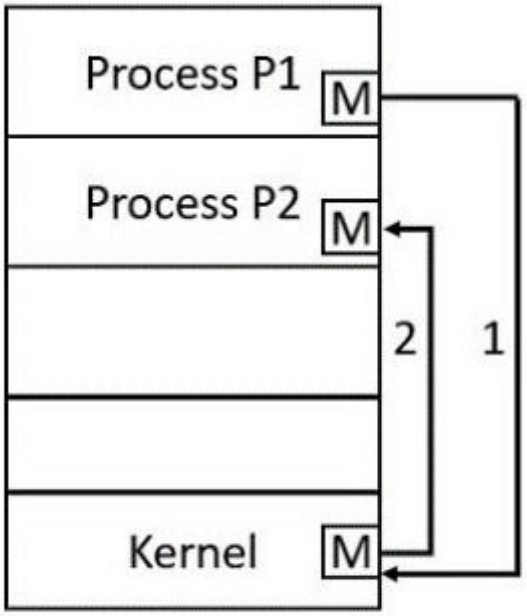
۱. ارسال پیام

[2] Inter-Process Communication (IPC)
[3] Action
[4] Synchronize

۲. دریافت پیام

پیام‌هایی که توسط پردازه‌ها ارسال می‌شوند، هم می‌توانند سایز ثابت و هم سایز متغیر داشته باشند. برای پیام‌های با سایز ثابت، پیاده‌سازی سطح سیستمی آن ساده است، ولی برنامه‌نویسی آن را پیچیده‌تر می‌کند. خلاف آن، برای پیام‌های با سایز متغیر، نیاز به پیاده‌سازی سطح سیستمی پیچیده‌تری داریم، در عین حال برنامه‌نویسی آن راحت‌تر است.

نکته: اگر دو پردازه P1 و P2 بخواهند با یک‌دیگر ارتباط برقرار کنند (یعنی هم پیام ارسال و هم پیام دریافت کنند)، به این معنی است که باید بین آن‌ها یک ارتباط[5] وجود داشته باشد.



Message Passing System

تفاوت‌های مهم:

- مدل حافظه‌ی مشترک برای ارتباط پردازه‌های بین سیستم‌های تک‌پردازنده‌ها و یا چند پردازنده است که در آن، همه‌ی پردازنده‌ها روی یک ماشین قرار دارند و می‌توانند از به اشتراک گذاشتن فضای آدرس مشترک ارتباط برقرار کنند؛ این در حالی است که مدل ارسال متن، در محیط‌های توزیع‌شده استفاده می‌شوند که در آن پردازه‌ها روی ماشین‌های مختلف با کمک شبکه می‌خواهند با یک‌دیگر ارتباط برقرار کنند.

[5] Link

- در مدل حافظه‌ی مشترک برای خواندن یا نوشتن داده، باید برنامه‌ای نوشته شود اما در مدل ارسال متن نیازی به نوشته‌شدن هیچ برنامه‌ای نیست.
- مدل حافظه‌ی مشترک، بیشینه‌ی سرعت محاسبات را دارد، زیرا از حافظه استفاده می‌کند که نیازی به فراخوانی سیستمی[6] ندارد (تنها فراخوانی سیستمی هنگام به‌وجود آمدن حافظه‌ی مشترک است)؛ اما مدل ارسال متن زمان‌بر است؛ زیرا از طریق فراخوانی سیستمی پیاده‌سازی شده است.

پیاده‌سازی مدل ارسال متن از طریق مدل حافظه‌ی مشترک:

برای پیاده‌سازی می‌توانیم یک کانال FIFO مطمئن با استفاده از ثبات‌های تک‌نویسنده و تک‌خواننده و الگوریتم polling بسازیم. روی‌کرد ساده این‌طور است که برای هر یال uv که در مدل ارسال متن داریم، یک ثبات با اندازه بسیار بزرگ ruv بسازیم و u هر بار که می‌خواهد ارسال جدیدی را انجام دهد، کل دنباله هر پیامی که تا به حال به v ارسال کرده است، روی ruv می‌نویسد. برای دریافت پیام‌ها، v همه‌ی ثبات‌های ورودی را به‌طور دوره‌ای بررسی می‌کند و همه‌ی پیام‌هایی را که قبلا پردازش نکرده باشد را تحویل می‌دهد.

این روش می‌تواند با اضافه کردن سازوکار acknowledgment در یک ثبات دیگر به نام ack(vu) بهبود پیدا کند؛ به‌طوری که u هر بار فقط یک پیام برای ruv ارسال می‌کند و پیام‌های بعدی را تا زمانی که v در ack(vu) بنویسد که پیام در ruv دریافت شده است در صف قرار می‌دهد. با کمی تغییرات می‌توان ruv را تنها به سه حالت ممکن (ارسال ۰، ارسال ۱ و ریست) و ack(vu) را به یک بیت تقلیل داد. غیر از حالتی که پردازه مرده ارسال و دریافت را متوقف می‌کند، خرابی پردازه‌ها هیچ تاثیری روی این پروتکل نمی‌گذارد.

پیاده‌سازی مدل حافظه‌ی مشترک از طریق مدل ارسال متن:

برای پیاده‌سازی n کپی از ثبات‌ها روی هر کدام از پردازه‌ها می‌گیریم. هر کپی پردازه یک زوج مرتب (مقدار و زمان) را در خود نگه می‌دارد که زمان هر مقدار صحیح نامحدود می‌تواند باشد. ابتدا همه با زوج مرتب (مقدار پیش‌فرض و ۰) شروع می‌کنند. پردازه مقدار کپی را با مقدار جدید (v و t) که از هر پردازه‌ی دیگر p دریافت کرده است، مشروط بر این‌که مقدار t بزرگ‌تر از مقدار زمان فعلی باشد، به‌روزرسانی می‌کند.

سپس به فرآیند p اطلاع می‌دهد که آیا مقدار کپی که روی خود داشته است را به‌روزرسانی کرده است یا خیر. این مقدار را با استفاده از ack(v,t) اطلاع می‌دهد. پردازه هم‌چنین به درخواست read(u) با استفاده از ack(value, timestamp, u) پاسخ می‌دهد. در این‌جا u یک مقدار برای تمایز بین عملیات‌های خواندن متفاوت است که در فرآیند خواندن اشتباهی صورت نگیرد.

[6] System call

حال برای نوشتن یک مقدار، نویسنده زمان خود را افزایش داده، مقدار خود را به‌روز رسانی کرده و سپس دستور write(value, timestamp) را برای همه‌ی پردازه‌های دیگر ارسال می‌نماید. عملیات نوشتن تنها زمانی پایان می‌یابد که نویسنده همه‌ی acknowledgement های حاوی مقدار زمانی جدید را از اکثر پردازه‌ها دریافت کرده باشد.

برای خواندن یک مقدار، خواننده دو مرحله کار انجام می‌دهد:

۱. دستور read(u) را به همه‌ی پردازه‌ها ارسال می‌کند (که در آن u هر مقداری است که قبلا استفاده نکرده است) و منتظر می‌ماند که acknowledgement ها را از اکثریت پردازه‌ها دریافت کند. مقدار مربوط به v با حداکثر زمان را برمی‌گرداند (بدون توجه به این‌که چه تعدادی از پردازه‌ها ارسال شده است).

۲. سپس دستور write(v, t) را به تمام فرآیندها می‌فرستد و منتظر پاسخ ack(v, t) از اکثر پردازه‌ها می‌ماند.

A channel-based coordination model for component composition:

مدل‌های مختلفی برای طراحی و ساخت نرم‌افزار وجود دارد. یکی از معروف‌ترین مدل‌هایی که وجود دارد، مدل طراحی modular است که در آن module ها، با اتکا به رابط‌های یک‌دیگر، مانند تکه‌های پازل بهم متصل می‌شوند و ساختار نرم‌افزار را تشکیل می‌دهند. در مقابل طراحی modular، طراحی بر اساس اجزا[7]ی نرم‌افزار وجود دارد که در آن، اجزا کم‌تر به یک‌دیگر و محیطی که در آن مستقر[8] می‌شوند وابسته هستند؛ اگر قرار است عمل‌کرد هر یک از این module ها توسط یک جزء پشتیبانی شود، بخش عمده‌ای از رابط باید از اجزای جداگانه حذف شود، زیرا مقررات [9] برای واسط یک جزء، به زمینه[10]ای که در آن و دیگر اجزایی که ممکن است با آن‌ها تعامل داشته باشد، مستقر شده، بستگی دارد. با توجه به این طراحی، چون بخش قابل توجهی از ساختار رابط‌ها با استفاده از کدهایی که اصطلاحا به آن‌ها «glue code» گفته می‌شود، پیاده‌سازی می‌شود، خود این کدها می‌توانند باعث مشکلاتی شوند که نگهداری از آن را بسیار سخت کند اما مزیت‌هایی دارد که به طراحی بسیار کمک می‌کند، زیرا اجزایی که قابلیت استفاده‌ی مکرر، نگهداری و جای‌گزینی را دارند را به یکدیگر متصل می‌کند. تا این‌جا فهمیدیم که طراحی بر اساس اجزاء با وجود مشکلات نگهداری‌ای که دارد، اما قابلیت زیادی را به دلیل خواص خود اجزا به ما می‌دهد (که بالاتر به آن اشاره شد).

یک روش جای‌گزین، نوشتن اسکریپتی است که glue code را به صورت ترکیبی با استفاده از اتصال‌دهنده[11]های اولیه می‌سازد. یک روش در این راستا این است که از کانال‌ها به عنوان مبانی اولیه‌ای استفاده کرد که از چنین اتصال‌دهنده‌هایی از آن‌ها ساخته شده‌اند. کانال یک رسانه‌ی ارتباطی نقطه‌به‌نقطه است که هویت منحصر به فرد با دو طرف[12] متمایز دارد. مدل‌های کانال‌محور «کامل» هستند، یعنی می‌توانند به‌راحتی می‌توانند مدل‌های اولیه را از مدل‌های ارتباطی دیگر (مانند مدل حافظه‌ی مشترک، ارسال متن و یا فراخوانی رویه از راه دور[13]) مدل‌سازی کند.

مزایای مدل‌های کانال محور عبارتند از:

۱. **کارایی**: از ارتباط نقطه‌به‌نقطه پشتیبانی می‌کنند.

---

[7] Component
[8] Deploy
[9] Provision
[10] Context
[11] Connector
[12] End
[13] Remote procedure call

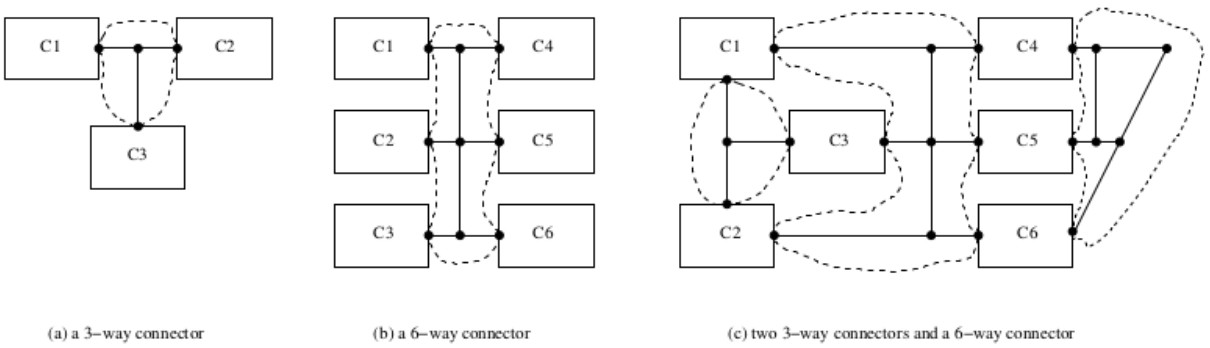
۲. **امنیت:** خلاف مدل حافظه‌ی مشترک، در ارتباط نقطه‌به‌نقطه، یک حفاظ ارتباطی را برای جلوگیری از سهوا در معرض قرار گرفتن اطلاعات توسط طرفین سوم مدل می‌کند.

۳. **روشن‌بودن معماری**: در مقایسه با معماری‌های دیگر از روی معماری می‌توان اطلاعات مرتبط به ارتباط اجزا و موجودیت‌ها با یک‌دیگر را دید.

۴. **ناشناس بودن** : خلاف مدل فراخوانی رویه‌ها از راه دور و مدل ارسال متن، ارتباط‌های کانال‌محور از ارتباطات ناشناس پشتیبانی می‌کنند.

Reo براساس محاسباتی از کانال‌ها است که در آن اتصال‌دهنده‌های پیچیده از طریق ترکیب اتصال‌دهنده‌های ساده‌تر درست می‌شوند. از Reo می‌توان به عنوان «glue code» در طراحی‌های جزءمحور استفاده کرد که در آن سیستم از طریق ترکیب اجزا که با یک‌دیگر به‌صورت ناشناس با استفاده از اتصال‌دهنده‌های Reo تعامل و هم‌کاری می‌کنند ساخته می‌شود.

اجزای Reo عبارتند از: ۱) نمونه اجزا (مستطیل‌ها) ۲) اتصال‌دهنده‌ها (خط‌ها) ۳) کانال (نقطه‌چین‌ها)



(a) a 3–way connector (b) a 6–way connector (c) two 3–way connectors and a 6–way connector

هر کانال در مدل Reo هویت منحصر به فرد دارد و جهت ندارد. اما هر کانال دارای دقیقا دو طرف جهت‌دار است. منبع[14]، از جایی که داده‌ها وارد کانال می‌شوند و مقصد[15] جایی است که داده‌ها از کانال خارج می‌شوند. در مدل Reo هم اجزاء و هم کانال‌ها فرض می‌شوند که سیار[16] هستند. یعنی می‌توانند از یک مکان به مکان دیگری در طول حیات خود حرکت کنند. هنگامی که این اتفاق می‌افتد، طرفین کانال (منبع و مقصد)، هم‌چنان متصل باقی می‌مانند. با توجه به تعاریفی که از راجع‌به مدل کانال‌محور ارائه شد، عملیاتی که روی کانال‌ها انجام می‌شوند شامل موارد زیر است:

- Create
- Forget

[14] Source
[15] Sink
[16] Mobile

- Move
- Connect
- Wait
- Read
- Take
- Write

Reo از الگوهایی برای منظم کردن عملیات ورودی و خروجی کانال استفاده می‌کند. الگو یک عبارتی است که با یک مورد از داده‌ها زمانی که در حال نوشته شدن، یا خوانده شدن از و یا جریان از طریق کانال است، تطابق می‌یابد. در این الگوریتم دو نوع کانال داریم:

۱. هم‌گام: اگر موفقیت جفت عملیات در دو سمت کانال را تا زمانی که فقط به‌طور هم‌زمان موفق شوند، به تاخیر بیاندازد.

۲. غیر هم‌گام: خلاف مورد بالا.

در نهایت برای جمع‌بندی، مدل Reo که یکی از مدل‌های کانال‌محور است به ما کمک می‌کند تا نرم‌افزارهای خود را در سیستم‌های توزیع‌شده را طوری طراحی کنیم که نگهداری آن آسان‌تر باشد، منعطف‌تر باشد و اضافه‌کردن ویژگی‌ها و تکامل آن راحت‌تر باشد.