HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG

KHOA ĐÀO TẠO SAU ĐẠI HỌC

BÀI TẬP
CÁC HỆ THỐNG PHÂN TÁN

CHƯƠNG 1

Giảng viên: **TS. Kim Ngọc Bách**

Lớp: **M24CQHT02-B**

Nhóm: **14**

Danh sách học viên: **B24CHHT076 – Nguyễn Thanh Hùng**
B24CHHT078 – Quản Trường Huy
B24CHHT092 – Tô Thanh Thái

Hà Nội – 2025

1 Câu 1

1.1 Nêu và giải thích hai đặc điểm quan trọng nhất của hệ thống phân tán.

Hai đặc điểm cốt lõi làm nên bản chất của một hệ thống phân tán là: hệ thống bao gồm các phần tử tính toán tự trị, và hệ thống xuất hiện như một thực thể duy nhất và nhất quán đối với người dùng.

1.1.1 Tập hợp các phần tử tính toán tự chủ

Hệ thống phân tán được cấu thành từ nhiều phần tử tính toán tự chủ (nodes), có thể là các máy tính hiệu năng cao hoặc các thiết bị nhỏ hơn. Mỗi node hoạt động độc lập, nhưng để tạo thành một hệ thống phân tán hữu ích, chúng cần phối hợp để đạt được mục tiêu chung thông qua việc trao đổi thông điệp. Một node sẽ phản ứng với các thông điệp đến, xử lý chúng và tạo ra các thông điệp mới để tiếp tục giao tiếp.

Do tính độc lập của các node, mỗi node sẽ có khái niệm riêng về thời gian, không có một "đồng hồ toàn cục" chung. Điều này đặt ra những thách thức lớn về đồng bộ hóa và phối hợp.

Việc quản lý tư cách thành viên và tổ chức của tập hợp các node là rất quan trọng. Có sự phân biệt giữa nhóm mở (open groups), nơi bất kỳ node nào cũng có thể tham gia và giao tiếp, và nhóm đóng (closed groups), nơi chỉ các thành viên được phép giao tiếp và cần một cơ chế riêng để tham gia hoặc rời nhóm.

Kiểm soát truy cập trong hệ thống phân tán rất phức tạp. Đầu tiên, cần có cơ chế xác thực node. Thứ hai, mỗi node phải kiểm tra xem nó có đang giao tiếp với một thành viên hợp lệ hay không, tránh trường hợp kẻ xâm nhập. Cuối cùng, nếu tính bảo mật thông tin là yếu tố quan trọng, thì việc một thành viên có thể dễ dàng giao tiếp với các phi thành viên sẽ đặt ra các vấn đề về lòng tin.

Về mặt tổ chức, hệ thống phân tán thường được tổ chức dưới dạng mạng overlay. Trong mạng overlay, một node thường là một tiến trình phần mềm có danh sách các tiến trình khác mà nó có thể gửi tin nhắn trực tiếp. Có hai loại mạng overlay chính:

- Overlay có cấu trúc: Mỗi node có một tập hợp các hàng xóm được xác định rõ ràng (ví dụ: được tổ chức theo cây hoặc vòng logic).
- Overlay không cấu trúc: Mỗi node có các tham chiếu đến các node khác được chọn ngẫu nhiên.

Trong mọi trường hợp, mạng overlay cần luôn được kết nối, đảm bảo rằng luôn có một đường truyền thông cho phép các node định tuyến tin nhắn giữa chúng. Mạng ngang hàng (P2P) là một ví dụ điển hình của mạng overlay. Việc tổ chức các node trong hệ thống phân tán đòi hỏi nỗ lực đáng kể và thường là một trong những phần phức tạp nhất của việc quản lý hệ thống phân tán.

1.1.2 Xuất hiện như một hệ thống nhất quán duy nhất

Một đặc điểm quan trọng khác của hệ thống phân tán là nó phải xuất hiện như một hệ thống thống nhất duy nhất (single coherent system). Mặc dù các thành phần của hệ thống phân tán được phân tán trên mạng máy tính, người dùng cuối không nên nhận thấy sự phân tán đó. Điều này có nghĩa là hệ thống phải hoạt động theo mong đợi của người dùng, bất kể việc tương tác diễn ra ở đâu, khi nào và như thế nào.

Để đạt được sự thống nhất này là một thách thức lớn. Ví dụ, người dùng không nên biết chính xác tiến trình đang chạy trên máy tính nào, hay dữ liệu được lưu trữ ở đâu, hoặc việc hệ thống có đang sao chép dữ liệu để tăng hiệu suất hay không. Khái niệm này được gọi là tính trong suốt phân tán (distribution transparency), một mục tiêu thiết kế quan trọng trong các hệ thống phân tán. Nó tương tự như cách các hệ điều hành giống Unix ẩn đi sự khác biệt giữa các tài nguyên (file, thiết bị lưu trữ, bộ nhớ chính, mạng) thông qua một giao diện hệ thống file thống nhất.

Tuy nhiên, việc theo đuổi một hệ thống thống nhất duy nhất tạo ra một sự đánh đổi quan trọng. Không thể bỏ qua thực tế rằng hệ thống phân tán bao gồm nhiều node được nối mạng, và do đó, tại bất kỳ thời điểm nào, chỉ một phần của hệ thống có thể gặp lỗi. Điều này dẫn đến các hành vi không mong muốn, chẳng hạn như một số ứng dụng vẫn có thể tiếp tục thực thi thành công trong khi những ứng dụng khác bị ngừng hoạt động đột ngột. Mặc dù lỗi cục bộ là cố hữu trong bất kỳ hệ thống phức tạp nào, nhưng trong hệ thống phân tán, chúng đặc biệt khó che giấu.

1.2 Trình bày ba lý do cơ bản khiến các ứng dụng phân tán phức tạp hơn so với các ứng dụng đơn lẻ.

1.2.1 Độ trễ mạng đáng kể

Trong hệ thống phân tán, thời gian truyền thông giữa các tiến trình có thể lên tới hàng trăm mili giây – chậm hơn rất nhiều so với giao tiếp trong cùng một máy. Việc giả định mạng có độ trễ thấp dẫn đến các thiết kế đồng bộ không hiệu quả và dễ gây nghẽn. Ứng dụng đơn lẻ không gặp vấn đề này do mọi giao tiếp diễn ra gần như tức thì.

1.2.2 Mạng không đáng tin cậy

Giao tiếp mạng có thể gặp lỗi như mất gói, trễ, trùng lặp hoặc lỗi phần mềm/hardware tại các node. Do đó, ứng dụng phân tán cần thêm các cơ chế như retry, acknowledgment, hoặc timeout để đảm bảo độ tin cậy. Trong khi đó, ứng dụng đơn lẻ hiếm khi cần xử lý các loại lỗi phức tạp như vậy.

1.2.3 Tính không đồng nhất

Hệ thống phân tán gồm nhiều máy với kiến trúc, hệ điều hành, chuẩn dữ liệu khác nhau. Điều này đòi hỏi ứng dụng phải xử lý vấn đề tương thích, ví dụ như sự khác biệt về byte order (big endian vs little endian) hoặc giao tiếp giữa các môi trường thực thi

khác nhau. Ứng dụng đơn lẻ không gặp khó khăn này vì chỉ chạy trong một môi trường đồng nhất.

2 Câu 2

2.1 Phân tích các yếu tố phần cứng (CPU, bộ nhớ, kênh truyền) và yếu tố mạng (băng thông, topology) ảnh hưởng đến hiệu năng hệ thống phân tán.

2.1.1 Yếu tố phần cứng

2.1.1.1 CPU

CPU là bộ não của mọi tiến trình tính toán. Đối với các dịch vụ nặng về tính toán (compute-bound), như dịch vụ tính toán lộ trình tối ưu có tính đến giao thông thời gian thực, CPU là yếu tố giới hạn chính. Nếu số lượng yêu cầu tăng lên vượt quá khả năng xử lý của CPU đơn lẻ, ngay cả một hệ thống cao cấp cũng sẽ trở nên quá tải.

Ví dụ: Một dịch vụ cần hàng chục giây để hoàn thành một yêu cầu do tính toán phức tạp sẽ nhanh chóng gặp tắc nghẽn CPU nếu có quá nhiều yêu cầu đồng thời. Mặc dù có thể mở rộng bằng cách thêm nhiều CPU hoặc máy chủ, nhưng nếu chỉ tập trung vào một máy chủ duy nhất, CPU sẽ là điểm nghẽn.

2.1.1.2 Bộ nhớ

Bộ nhớ chính đóng vai trò quan trọng trong việc lưu trữ dữ liệu và chương trình đang được xử lý. Khi một dịch vụ cần xử lý lượng lớn dữ liệu (ví dụ: một công cụ tìm kiếm cần so khớp truy vấn với toàn bộ tập dữ liệu), nếu dữ liệu đó vượt quá dung lượng bộ nhớ chính của máy, hệ thống sẽ phải liên tục truy cập vào ổ đĩa.

Ví dụ: Một công cụ tìm kiếm tập trung được thiết kế kém có thể phải xử lý lượng dữ liệu vượt quá RAM. Điều này khiến thời gian xử lý bị chi phối bởi các truy cập đĩa chậm hơn nhiều so với truy cập bộ nhớ. Dù có kỹ thuật lập chỉ mục tiên tiến, việc phải đọc dữ liệu từ đĩa vẫn là một nút thắt cổ chai lớn.

2.1.1.3 Kênh truyền (Disk I/O - Input/Output của đĩa)

Kênh truyền, đặc biệt là tốc độ truy cập và truyền dữ liệu giữa ổ đĩa và bộ nhớ chính (Disk I/O), là yếu tố giới hạn quan trọng đối với các dịch vụ nặng về I/O (I/O-bound). Khi dữ liệu cần xử lý lớn hơn RAM, hệ thống sẽ phải liên tục đọc/ghi từ đĩa. Tốc độ ổ đĩa (HDD so với SSD), giao diện kết nối (SATA, SAS, NVMe) và RAID cấu hình đều ảnh hưởng trực tiếp đến hiệu năng I/O.

Ví dụ: Tiếp tục với ví dụ công cụ tìm kiếm, nếu dịch vụ phải liên tục đọc lượng lớn dữ liệu từ đĩa, hiệu năng sẽ bị giảm sút nghiêm trọng. Đơn giản việc thêm nhiều đĩa hoặc đĩa tốc độ cao hơn không phải là giải pháp bền vững khi số lượng yêu cầu và kích

thước dữ liệu tiếp tục tăng, bởi vì vẫn có giới hạn về băng thông I/O của toàn bộ hệ thống.

2.1.2 Yếu tố mạng

2.1.2.1 Băng thông

Băng thông là yếu tố giới hạn chính đối với các dịch vụ truyền tải dữ liệu lớn, đặc biệt là dịch vụ truyền tải dữ liệu đa phương tiện hoặc các hệ thống cần di chuyển lượng lớn dữ liệu giữa các node. Nếu băng thông của đường truyền đi (outgoing transmission lines) bị bão hòa, dịch vụ sẽ không thể đáp ứng được các yêu cầu mới hoặc sẽ gặp tình trạng giật lag, độ trễ cao.

Ví dụ: Một dịch vụ video theo yêu cầu (video-on-demand) cần truyền video chất lượng cao đến nhiều người dùng. Một luồng video có thể yêu cầu băng thông 8 đến 10 Mbps. Nếu dịch vụ thiết lập các kết nối điểm-điểm với hàng ngàn khách hàng, nó sẽ nhanh chóng đạt đến giới hạn băng thông của đường truyền đi của chính mình, dẫn đến tắc nghẽn và trải nghiệm người dùng kém.

2.1.2.2 Độ trễ

Độ trễ cao có thể làm giảm hiệu quả giao tiếp giữa các node, đặc biệt là trong các giao thức yêu cầu nhiều vòng trao đổi tin nhắn (round trips) hoặc các ứng dụng nhạy cảm với thời gian thực.

2.1.2.3 Topology (Cấu trúc liên kết mạng):

Cấu trúc liên kết mạng mô tả cách các node được kết nối với nhau. Một topology được thiết kế tốt có thể giảm thiểu độ trễ, tăng cường băng thông hiệu quả và cải thiện khả năng chịu lỗi. Ví dụ, trong một hệ thống phân tán, việc các node gần nhau về mặt địa lý (trong cùng một trung tâm dữ liệu) sẽ có độ trễ thấp hơn so với việc chúng được đặt ở các khu vực địa lý khác nhau. Các cấu trúc liên kết mạng như hình sao, bus, mesh, hoặc tree đều có ưu nhược điểm riêng về khả năng mở rộng, độ tin cậy và chi phí. Trong bối cảnh hệ thống phân tán, các mạng overlay (như P2P) là một dạng topology logic được xây dựng trên nền tảng mạng vật lý, cho phép các node giao tiếp trực tiếp với nhau dựa trên một cấu trúc được định nghĩa (có cấu trúc hoặc không cấu trúc).

Ví dụ: Trong một hệ thống lưu trữ phân tán, nếu các bản sao dữ liệu được phân tán quá xa nhau, thời gian đồng bộ hóa và truy cập sẽ tăng lên đáng kể, ảnh hưởng đến hiệu năng tổng thể. Một topology mạng tối ưu sẽ cố gắng giữ các node thường xuyên giao tiếp gần nhau nhất có thể.

2.2 Tại sao hệ điều hành phân tán (distributed OS) và hệ điều hành mạng (network OS) lại có yêu cầu khác nhau về quản lý tài nguyên?

Trong hệ thống phân tán, việc quản lý tài nguyên là một thách thức lớn do tài nguyên không chỉ nằm trên một máy tính mà được phân tán trên nhiều node khác nhau. Hai mô hình phổ biến – Hệ điều hành mạng (Network Operating System - NOS) và Hệ

điều hành phân tán (Distributed Operating System - DOS) – có cách tiếp cận rất khác nhau đối với việc trừu tượng hóa và điều phối các tài nguyên này. Sự khác biệt nằm ở mức độ tích hợp, tính minh bạch, và chiến lược quản lý tài nguyên.

2.2.1 NOS – Hệ điều hành mạng

NOS bao gồm nhiều máy tính độc lập, mỗi máy chạy hệ điều hành riêng và quản lý tài nguyên cục bộ. Các đặc điểm chính:

- Quản lý tài nguyên cục bộ: Mỗi máy tự chịu trách nhiệm quản lý CPU, bộ nhớ, ổ đĩa… của riêng nó. Người dùng phải biết tài nguyên ở đâu để sử dụng.
- Minh bạch phân tán thấp: Người dùng nhìn thấy sự phân tán rõ ràng: biết rõ tài nguyên nằm ở máy nào, đường dẫn tệp ra sao, và phải sử dụng các cơ chế giao tiếp mạng như socket hoặc FTP.
- Không đồng nhất bị phơi bày: Do chạy trên các hệ điều hành và phần cứng khác nhau, lập trình viên hoặc người dùng phải xử lý sự khác biệt về biểu diễn dữ liệu, cách truy cập tài nguyên…
- Giao tiếp tường minh: Tiến trình giao tiếp phải gửi/nhận dữ liệu rõ ràng qua mạng, xử lý lỗi kết nối và đồng bộ thủ công.
- Chịu lỗi thấp: Nếu một máy bị lỗi, tài nguyên tương ứng không truy cập được và lỗi thường hiển thị trực tiếp đến người dùng mà không có lớp che chắn.

NOS phù hợp trong các môi trường đơn giản, không yêu cầu tính tích hợp cao giữa các node.

2.2.2 DOS – Hệ điều hành phân tán

Ngược lại, DOS cố gắng tạo ảo ảnh về một hệ điều hành duy nhất đang quản lý toàn bộ hệ thống phân tán. Những đặc điểm chính bao gồm:

- Quản lý tài nguyên toàn cục: Tài nguyên từ nhiều máy được hợp nhất và quản lý thống nhất, giúp hệ thống tự động phân phối và cân bằng tải.
- Minh bạch phân tán cao: Người dùng không cần biết tài nguyên nằm ở đâu, được sao chép hay di chuyển khi nào. Các dạng minh bạch bao gồm vị trí, truy cập, sao chép, lỗi, di chuyển và đồng thời.
- Xử lý không đồng nhất: Sử dụng các kỹ thuật như ảo hóa và middleware để cung cấp môi trường thống nhất bất chấp sự đa dạng của nền tảng phần cứng và hệ điều hành.
- Giao tiếp cấp cao: Giao tiếp qua RPC, message queues, hoặc các API trừu tượng giúp giảm gánh nặng lập trình và xử lý lỗi.
- Đồng bộ hóa và chịu lỗi: Cung cấp cơ chế đồng bộ (đồng hồ logic, loại trừ lẫn nhau phân tán), phục hồi sau lỗi, tái khởi động tiến trình… để hệ thống có thể tiếp tục hoạt động ngay cả khi một phần bị lỗi.
- Hỗ trợ đa miền quản trị: Trong môi trường cloud hoặc grid, DOS còn phải xử lý chính sách và bảo mật giữa các tổ chức khác nhau.

- Tập trung vào khả năng mở rộng: DOS được thiết kế để dễ dàng mở rộng về mặt số lượng node, phạm vi địa lý, và quy mô tài nguyên.

3 Câu 3

3.1 Nêu và so sánh ba loại hệ thống phân tán: điện toán phân tán, thông tin phân tán và lan tỏa phân tán.

3.1.1 Điện toán phân tán hiệu năng cao (High Performance Distributed Computing)

Mục tiêu chính: Đây là loại hệ thống phân tán được sử dụng cho các tác vụ điện toán hiệu năng cao.

Nguồn gốc: Thường xuất phát từ lĩnh vực điện toán song song.

Các phân nhóm chính:

- Điện toán cụm: Gồm một tập hợp các máy trạm hoặc PC tương tự nhau, được kết nối chặt chẽ bằng mạng cục bộ tốc độ cao. Mỗi nút chạy cùng một hệ điều hành. Các cụm hiện đại đang chuyển sang giải pháp lai (hybrid), trong đó middleware được phân vùng chức năng trên các nút khác nhau. Điều này giúp các nút tính toán với hệ điều hành chuyên dụng, gọn nhẹ cung cấp hiệu suất tối ưu cho các ứng dụng tính toán chuyên sâu, đồng thời các nút được cấu hình đặc biệt có thể xử lý tối ưu chức năng lưu trữ (ví dụ: máy chủ tệp và thư mục) và các dịch vụ middleware chuyên dụng khác.
- Điện toán lưới: Ban đầu tập trung mạnh vào chia sẻ tài nguyên trên toàn thế giới, sau đó đã dẫn đến khái niệm điện toán đám mây.
- Điện toán đám mây: Được tổ chức thành bốn lớp chính:
  - Phần cứng: Lớp thấp nhất, quản lý bộ xử lý, bộ định tuyến, hệ thống điện và làm mát. Thường được triển khai tại các trung tâm dữ liệu và khách hàng không trực tiếp nhìn thấy tài nguyên này.
  - Cơ sở hạ tầng: Xương sống của hầu hết các nền tảng điện toán đám mây. Triển khai kỹ thuật ảo hóa để cung cấp cho khách hàng cơ sở hạ tầng gồm các tài nguyên lưu trữ và tính toán ảo. Mọi thứ xoay quanh việc phân bổ và quản lý các thiết bị lưu trữ ảo và máy chủ ảo.
  - Nền tảng: (Tên lớp được nhắc đến trong cấu trúc 4 lớp).
  - Phần mềm: (Tên lớp được nhắc đến trong cấu trúc 4 lớp).

3.1.2 Hệ thống thông tin phân tán (Distributed Information Systems)

Mục tiêu chính: Tập trung vào tích hợp ứng dụng doanh nghiệp (enterprise application integration - EAI), cho phép các ứng dụng giao tiếp trực tiếp với nhau.

Ứng dụng điển hình: Các hệ thống này thường hỗ trợ các hệ thống phân tán được tìm thấy trong môi trường văn phòng truyền thống, nơi cơ sở dữ liệu đóng vai trò quan trọng.

Xử lý giao dịch phân tán: Các hoạt động trên cơ sở dữ liệu được thực hiện dưới dạng giao dịch.

Thuộc tính ACID của giao dịch: Các giao dịch tuân thủ bốn thuộc tính sau:

- Atomic (Nguyên tố): Đối với thế giới bên ngoài, giao dịch xảy ra không thể phân chia (tất cả hoặc không gì cả).
- Consistent (Nhất quán): Giao dịch không vi phạm các bất biến của hệ thống.
- Isolated (Cô lập): Các giao dịch đồng thời không can thiệp vào nhau.
- Durable (Bền vững): Một khi giao dịch đã cam kết, các thay đổi là vĩnh viễn.

Giám sát xử lý giao dịch (TP monitor): Là thành phần cốt lõi trong việc tích hợp các ứng dụng ở cấp máy chủ hoặc cơ sở dữ liệu. Nhiệm vụ chính là cho phép một ứng dụng truy cập nhiều máy chủ/cơ sở dữ liệu bằng cách cung cấp mô hình lập trình giao dịch và phối hợp cam kết các giao dịch con.

3.1.3 Hệ thống lan tỏa (Pervasive Systems)

Mục tiêu chính: Các thành phần nhỏ, hệ thống được cấu tạo theo kiểu tùy biến (ad hoc) và không còn được quản lý thông qua quản trị viên hệ thống.

Đặc điểm: Các yếu tố của nó đã lan rộng khắp nhiều phần trong môi trường của chúng ta.

Các loại chính: Bao gồm hệ thống điện toán toàn diện (ubiquitous computing systems), hệ thống di động (mobile systems) và mạng cảm biến (sensor networks).

- Hệ thống điện toán toàn diện:
  - Là hệ thống lan tỏa và hiện diện liên tục, người dùng tương tác liên tục với hệ thống mà thường không nhận thức được sự tương tác đang diễn ra.
  - Yêu cầu cốt lõi:
    - Phân tán: Các thiết bị được nối mạng, phân tán và truy cập một cách minh bạch.
    - Tương tác: Tương tác giữa người dùng và thiết bị rất kín đáo.
    - Nhận biết ngữ cảnh: Hệ thống nhận biết ngữ cảnh của người dùng để tối ưu hóa tương tác. Ngữ cảnh thường được đặc trưng bởi vị trí, danh tính, thời gian và hoạt động (ở đâu, ai, khi nào và làm gì).
    - Tự chủ: Các thiết bị hoạt động tự chủ không cần sự can thiệp của con người, tự quản lý cao.
    - Thông minh: Hệ thống tổng thể có thể xử lý nhiều hành động và tương tác động.

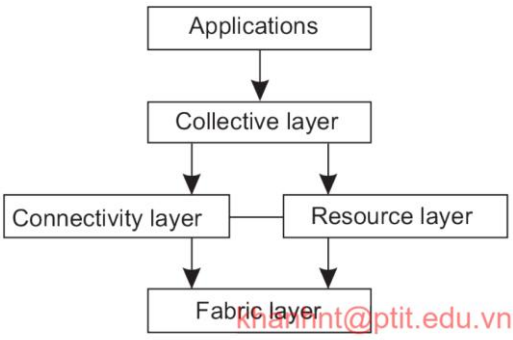
- Hệ thống di động: Chẳng hạn như điện thoại di động, hành vi di chuyển của con người có thể dự đoán được một cách đáng ngạc nhiên.
- Mạng cảm biến:
  - Thường là một phần của công nghệ hỗ trợ tính lan tỏa.
  - Các nút cảm biến thường hợp tác để xử lý dữ liệu được cảm biến một cách hiệu quả theo ứng dụng cụ thể.
  - Cần các tiện ích để xử lý dữ liệu trong mạng (in-network data processing) để tránh lãng phí tài nguyên và năng lượng.

3.2 Phân tích các lớp chính (application, middleware, resource) trong kiến trúc điện toán lưới và vai trò của từng lớp.

3.2.1 Đặc điểm chung

- Không giống như các hệ thống cluster truyền thống có tính đồng nhất (cùng hệ điều hành, phần cứng), điện toán lưới có tính đa dạng cao về phần cứng, hệ điều hành, chính sách bảo mật, v.v.
- Tài nguyên từ nhiều tổ chức được tập hợp lại để phục vụ cho tổ chức ảo.
- Các tài nguyên có thể bao gồm: máy tính hiệu năng cao, kho lưu trữ, cơ sở dữ liệu, hoặc các thiết bị mạng đặc biệt như kính viễn vọng hay cảm biến.

3.2.2 Kiến trúc phân lớp



3.2.2.1 Fabric Layer

- Giao diện truy cập tài nguyên cục bộ tại mỗi site.
- Hỗ trợ truy vấn trạng thái, khả năng và quản lý tài nguyên (vd: khóa, mở khóa).

3.2.2.2 Connectivity Layer

- Giao thức truyền thông và bảo mật để liên kết các tài nguyên từ xa.
- Hỗ trợ xác thực, ủy quyền – nơi chương trình thay mặt người dùng được cấp quyền.
- Cần thiết để truy cập hoặc truyền dữ liệu giữa các node.

3.2.2.3 Resource Layer

- Quản lý từng tài nguyên đơn lẻ.
- Gọi giao diện từ lớp fabric, sử dụng chức năng xác thực từ lớp connectivity.

- Thực hiện thao tác như lấy cấu hình, tạo tiến trình, truy xuất dữ liệu.

3.2.2.4 Collective Layer

- Điều phối truy cập nhiều tài nguyên cùng lúc.
- Cung cấp dịch vụ: khám phá tài nguyên, lập lịch, phân phối công việc, nhân bản dữ liệu...
- Không có chuẩn giao thức cố định, phản ánh phạm vi rộng lớn của các dịch vụ.

3.2.2.5 Application Layer

- Chứa các ứng dụng thực tế hoạt động trên môi trường lưới, dùng các lớp bên dưới để xử lý phân tán.

3.2.3 Tổng hợp

- Ba lớp Collective, Connectivity và Resource tạo thành middleware lưới.
- Kiến trúc lưới có tính phân quyền: mỗi site (đơn vị quản trị) vận hành độc lập.
- Xu hướng hiện nay chuyển sang kiến trúc dịch vụ (SOA) với Open Grid Services Architecture (OGSA), sử dụng chuẩn Web Services để truy cập tài nguyên.

4 Câu 4

4.1 Giải thích tại sao "tính sẵn sàng" (availability) được xem là mục tiêu quan trọng nhất của hệ thống phân tán.

Tính sẵn sàng (availability) là một mục tiêu thiết kế cốt lõi trong các hệ thống phân tán, thể hiện khả năng hệ thống có thể hoạt động và phản hồi đúng đắn tại bất kỳ thời điểm nào. Đây là một trong bốn yếu tố cấu thành nên một hệ thống đáng tin cậy (dependable), bên cạnh độ tin cậy, an toàn và khả năng bảo trì.

Trong môi trường phân tán, nơi các lỗi phần cứng, phần mềm và mạng là không thể tránh khỏi, tính sẵn sàng đặc biệt quan trọng để đảm bảo hệ thống vẫn phục vụ người dùng. Khả năng chịu lỗi – thông qua các cơ chế như nhóm tiến trình – thường được thiết kế để duy trì tính sẵn sàng ngay cả khi một số thành phần bị lỗi. Đây cũng là một thuộc tính sống còn, đảm bảo hệ thống "cuối cùng" vẫn đáp ứng yêu cầu.

Định lý CAP nhấn mạnh rằng trong trường hợp xảy ra phân vùng mạng, không thể đảm bảo đồng thời cả tính nhất quán và tính sẵn sàng. Nhiều hệ thống thực tế chọn ưu tiên tính sẵn sàng, chấp nhận sự không nhất quán tạm thời để duy trì hoạt động liên tục.

Mặc dù tính minh bạch phân tán hướng tới việc che giấu sự phân tán và lỗi khỏi người dùng, trên thực tế, điều này rất khó đạt được. Thay vào đó, duy trì một hệ thống luôn sẵn sàng, ngay cả khi có sự cố xảy ra, là mục tiêu thực tiễn và quan trọng hơn trong thiết kế hệ thống phân tán.

4.2 Nêu và so sánh ba hình thức "tính trong suốt" (trong suốt về truy nhập, vị trí và lỗi), và ví dụ đơn giản minh họa mỗi loại.

4.2.1 Tính trong suốt về truy nhập

Che giấu sự khác biệt trong cách biểu diễn dữ liệu và phương thức truy cập đối tượng. Người dùng không cần quan tâm đến định dạng dữ liệu hay giao thức truy cập cụ thể.

Ví dụ: Hệ điều hành Unix cung cấp một giao diện hệ thống tệp thống nhất, cho phép truy cập cả tệp cục bộ và tệp qua mạng thông qua cùng một cú pháp như /home/user/file.txt.

4.2.2 Tính trong suốt về vị trí

Ẩn vị trí vật lý nơi tài nguyên được lưu trữ. Người dùng tương tác thông qua tên logic, không phụ thuộc vào nơi đặt tài nguyên trong hệ thống.

Ví dụ: Một URL như http://www.example.com/page.html không tiết lộ nơi đặt máy chủ thật sự. Tài nguyên có thể di chuyển giữa các máy chủ mà người dùng không biết, miễn là URL vẫn hoạt động.

4.2.3 Tính trong suốt về lỗi

Ẩn sự cố xảy ra với các thành phần của hệ thống, giúp hệ thống tự động phục hồi mà người dùng không nhận biết được. Đây là dạng trong suốt khó đạt được nhất.

Ví dụ: Một dịch vụ web gặp lỗi máy chủ tạm thời sẽ tự động thử lại hoặc chuyển yêu cầu sang bản sao khác, đảm bảo người dùng vẫn nhận được phản hồi bình thường.

4.3 Trình bày mối quan hệ giữa "tính mở" (openness) và khả năng tương tác (interoperability) trong hệ thống phân tán.

4.3.1 Tính mở

Một hệ thống phân tán được coi là mở nếu nó cung cấp các thành phần có thể dễ dàng sử dụng bởi, hoặc tích hợp vào các hệ thống khác.

Đồng thời, một hệ thống phân tán mở thường sẽ bao gồm các thành phần có nguồn gốc từ nơi khác.

Để đạt được tính mở, các thành phần phải tuân thủ các quy tắc tiêu chuẩn mô tả cú pháp và ngữ nghĩa của các dịch vụ mà chúng cung cấp. Một cách tiếp cận chung để định nghĩa các dịch vụ này là thông qua các giao diện sử dụng ngôn ngữ định nghĩa giao diện (IDL).

4.3.2 Khả năng tương tác

Khả năng tương tác mô tả mức độ mà hai triển khai của các hệ thống hoặc thành phần từ các nhà sản xuất khác nhau có thể cùng tồn tại và hoạt động cùng nhau.

Điều này đạt được bằng cách chỉ dựa vào các dịch vụ của nhau như được quy định bởi một tiêu chuẩn chung.

4.3.3 Mối quan hệ giữa tính mở và khả năng tương tác

Tính mở là điều kiện tiên quyết để đạt được khả năng tương tác. Khi một hệ thống được thiết kế để mở, nó phải tuân thủ các tiêu chuẩn và quy tắc chung cho các giao diện dịch vụ của mình.

Khi các giao diện được chỉ định một cách chính xác – nghĩa là chúng đầy đủ và trung lập – thì một tiến trình bất kỳ cần một giao diện nhất định có thể "nói chuyện" với một tiến trình khác cung cấp giao diện đó.

Sự rõ ràng và tuân thủ tiêu chuẩn trong định nghĩa giao diện cho phép hai bên độc lập xây dựng các triển khai hoàn toàn khác nhau của cùng một giao diện, nhưng vẫn đảm bảo chúng hoạt động theo cùng một cách. Đây chính là cốt lõi của khả năng tương tác.

Do đó, tính mở thông qua việc chuẩn hóa và định nghĩa rõ ràng các giao diện của thành phần, trực tiếp dẫn đến khả năng tương tác giữa các thành phần và hệ thống khác nhau, bất kể chúng được phát triển bởi ai hoặc ở đâu.

Khả năng tương tác cũng được hỗ trợ bởi khả năng cấu hình hệ thống từ các thành phần khác nhau và dễ dàng thêm hoặc thay thế các thành phần mới mà không ảnh hưởng đến các thành phần hiện có (khả năng mở rộng). Các hệ thống nguyên khối thường khó thay thế hoặc điều chỉnh các thành phần mà không ảnh hưởng đến toàn bộ hệ thống, do đó chúng có xu hướng đóng thay vì mở.

5 Câu 5

5.1 So sánh ưu – nhược điểm của kiến trúc phân cấp và kiến trúc ngang hàng trong hệ thống phân tán.

5.1.1 Kiến trúc phân cấp

Kiến trúc phân cấp thường được thể hiện qua các mô hình như Client-Server hoặc các hệ thống có thành phần điều phối tập trung hoặc phân tầng rõ rệt, ví dụ như Hệ thống tên miền (DNS).

- Ưu điểm:
  - Đơn giản hóa quản lý và tương tác: Trong mô hình client-server, vai trò được phân định rõ ràng giữa máy khách (client) và máy chủ (server). Các giải pháp "client mỏng" (thin client), nơi phần lớn quá trình xử lý và lưu trữ dữ liệu được thực hiện ở phía máy chủ, làm cho việc quản lý hệ thống dễ dàng hơn đáng kể so với "client dày" (fat client).

- Tối ưu hóa hiệu suất cho các chức năng chuyên biệt: Trong các kiến trúc phân tầng hoặc đám mây, các nút tính toán có thể được cấu hình đặc biệt cho các ứng dụng chuyên sâu về tính toán, và các chức năng lưu trữ có thể được xử lý tối ưu bởi các máy chủ tệp và thư mục chuyên dụng. Mô hình Web truyền thống cũng là một ví dụ, nơi máy chủ xử lý các yêu cầu và tương tác với cơ sở dữ liệu.
- Khả năng mở rộng qua phân vùng và nhân bản có kiểm soát: Hệ thống có thể được phân vùng thành các phần nhỏ hơn và phân tán chúng khắp hệ thống. DNS là một ví dụ điển hình về việc phân cấp tên miền thành một cây các miền được tổ chức phân cấp, với mỗi vùng được quản lý bởi một máy chủ tên riêng. Các nút gốc của DNS được "phân tán cao và nhân bản" để đảm bảo hiệu suất và tính sẵn sàng.

- Nhược điểm:
  - Điểm nghẽn tiềm năng: Các dịch vụ tập trung có thể trở thành nút cổ chai khi lưu lượng yêu cầu tăng cao. Ví dụ, một dịch vụ tập trung có thể được mô hình hóa bằng lý thuyết xếp hàng, nơi việc sử dụng càng gần 1, tỷ lệ thời gian phản hồi trên thời gian dịch vụ càng tăng nhanh, khiến hệ thống gần như bị "đình trệ". Tương tự, một Home Agent trong MIPv6 có thể trở thành nút cổ chai nếu tất cả lưu lượng đều đi qua nó.
  - Điểm lỗi đơn (nếu không có nhân bản): Mặc dù các hệ thống lớn như DNS sử dụng nhân bản để chống lại điểm lỗi đơn, nhưng các hệ thống tập trung không được nhân bản đầy đủ vẫn có thể gặp rủi ro khi thành phần trung tâm gặp sự cố. Ví dụ, nếu máy chủ quản lý nhóm gặp sự cố, việc quản lý nhóm có thể ngừng hoạt động.
  - Khó khăn trong việc quản lý "fat clients": Việc có nhiều chức năng trên máy khách làm tăng nỗ lực cần thiết để đảm bảo phần mềm hoạt động ổn định và phụ thuộc vào nền tảng của máy khách, đòi hỏi việc duy trì nhiều phiên bản.

5.1.2 Kiến trúc ngang hàng

Kiến trúc ngang hàng là một dạng tổ chức hệ thống phân tán nơi "các tiến trình tạo thành một hệ thống ngang hàng đều bình đẳng". Trong mô hình này, các chức năng cần được thực hiện đều được đại diện bởi mọi tiến trình trong hệ thống.

Ưu điểm:

- Phân tán hoàn toàn và linh hoạt: Mỗi tiến trình hoạt động đồng thời như một máy khách và một máy chủ. Điều này loại bỏ sự phụ thuộc vào một điểm điều khiển tập trung, giúp hệ thống có khả năng phục hồi tốt hơn trước các lỗi của từng thành phần.
- Khả năng mở rộng tự nhiên và thích ứng với thay đổi: Các hệ thống ngang hàng được tổ chức dưới dạng mạng phủ (overlay network) và có thể bao gồm "một số lượng lớn các máy tính nối mạng". Chúng thường "có tính động cao, theo nghĩa

là các máy tính có thể tham gia và rời đi" liên tục. Các hệ thống P2P có cấu trúc như Chord cho phép tra cứu với chi phí logarit theo số lượng nút, thể hiện khả năng mở rộng tốt.
- Tận dụng tài nguyên phân tán: P2P là một cách hiệu quả để hỗ trợ chia sẻ tài nguyên bằng cách cho phép mỗi nút đóng góp và sử dụng tài nguyên.

Nhược điểm:

- Tính trong suốt kém hơn (trong một số trường hợp): Một số nhà nghiên cứu cho rằng nên có "ít tính trong suốt hơn", chẳng hạn bằng cách sử dụng giao tiếp kiểu tin nhắn rõ ràng hơn hoặc yêu cầu gửi và nhận kết quả từ các máy từ xa một cách rõ ràng hơn (ví dụ như cách Web tìm nạp trang). Điều này ngụ ý rằng các hệ thống P2P có thể phơi bày nhiều chi tiết phân tán hơn cho ứng dụng.
- Độ phức tạp trong thiết kế và quản lý: Do hành vi đối xứng và bản chất phân tán của các nút, việc tổ chức các tiến trình trong một mạng phủ là một thách thức. Các hệ thống P2P không cấu trúc duy trì danh sách các láng giềng một cách ngẫu nhiên, tạo thành một đồ thị ngẫu nhiên, điều này có thể làm cho việc tìm kiếm dữ liệu kém hiệu quả hơn so với các hệ thống có cấu trúc rõ ràng.
- Khó khăn trong việc xử lý lỗi cục bộ một cách trong suốt: "Các lỗi cục bộ ngăn cản việc dựa vào việc thực hiện thành công một dịch vụ từ xa". Điều này cho thấy việc đạt được tính trong suốt về lỗi hoàn toàn có thể rất khó khăn trong môi trường P2P, nơi các thành phần có thể thất bại độc lập.

5.2 Trình bày bốn mô hình hệ thống phân tán (phân tầng, đối tượng phân tán, kênh sự kiện, dữ liệu tập trung) và cho ví dụ ứng dụng điển hình cho mỗi mô hình.

5.2.1 Hệ thống phân tầng (Layered Systems)

Mô hình hệ thống phân tầng (Layered architectures) là một cách tổ chức phần mềm phổ biến, trong đó các thành phần được sắp xếp thành các lớp riêng biệt. Mỗi lớp cung cấp các dịch vụ cho lớp cao hơn và sử dụng dịch vụ của lớp thấp hơn.

Khái niệm:

- Trong mô hình phân tầng tiêu chuẩn, giao tiếp giữa các lớp chủ yếu là gọi xuống (downcalls), nghĩa là một lớp chỉ có thể gọi các hàm do lớp bên dưới cung cấp. Điều này tương tự như cách các giao thức truyền thông mạng được tổ chức thành các tầng (ví dụ: mô hình OSI).
- Một hệ thống phân tán có thể được chia thành ba cấp độ logic chính: cấp độ giao diện ứng dụng (application-interface level), cấp độ xử lý (processing level) và cấp độ dữ liệu (data level).

Ví dụ ứng dụng điển hình:

- Công cụ tìm kiếm Internet: Giao diện người dùng đơn giản (cấp độ giao diện ứng dụng) nhận các từ khóa, một cơ sở dữ liệu lớn các trang web được lập chỉ mục ở phần hậu kỳ (back end - cấp độ dữ liệu), và phần cốt lõi của công cụ tìm kiếm (xử lý truy vấn, xếp hạng kết quả) nằm ở cấp độ xử lý.
- Hệ thống hỗ trợ ra quyết định cho môi giới chứng khoán: Cũng có ba lớp tương tự: giao diện người dùng (front end), truy cập cơ sở dữ liệu tài chính (back end), và các chương trình phân tích ở giữa.
- Kiến trúc đa tầng: Đây là sự phân phối vật lý của các lớp logic trên nhiều máy khác nhau.
    - Kiến trúc hai tầng: Phổ biến nhất là máy khách chỉ chứa phần giao diện người dùng, còn máy chủ chứa phần xử lý và dữ liệu. Ví dụ, các ứng dụng ngân hàng nơi người dùng chuẩn bị giao dịch trên máy khách, sau đó tải lên cơ sở dữ liệu trên máy chủ ngân hàng để xử lý tiếp.
    - Kiến trúc ba tầng: Thêm một máy chủ riêng biệt cho lớp xử lý. Một ví dụ điển hình là TP monitor (transaction processing monitor) điều phối các giao dịch trên nhiều máy chủ/cơ sở dữ liệu khác nhau. Một ví dụ khác là các trang Web hiện đại, nơi một máy chủ Web chuyển yêu cầu đến một máy chủ ứng dụng để xử lý, và máy chủ ứng dụng này lại tương tác với máy chủ cơ sở dữ liệu.

5.2.2 Hệ thống đối tượng phân tán (Distributed Object Systems)

Mô hình này tập trung vào việc cho phép các đối tượng được phân tán trên nhiều máy tính có thể tương tác với nhau một cách minh bạch.

Khái niệm:

- Dựa trên ý tưởng về việc gọi phương thức từ xa (Remote Method Invocation - RMI) hoặc gọi thủ tục từ xa (Remote Procedure Call - RPC). Khi một đối tượng trên máy khách muốn gọi một phương thức trên một đối tượng từ xa trên máy chủ, nó không gọi trực tiếp mà thông qua một proxy phía máy khách.
- Proxy này có trách nhiệm đóng gói các tham số (marshaling) thành một thông điệp và gửi đến máy chủ.
- Ở phía máy chủ, một stub phía máy chủ (server stub), thường được gọi là skeleton, nhận thông điệp, giải nén các tham số và gọi phương thức thực tế trên đối tượng máy chủ. Kết quả sau đó được đóng gói lại và gửi về client qua stub và proxy.
- RPC/RMI che giấu sự phức tạp của việc truyền thông điệp, khiến việc gọi một thủ tục từ xa trông giống như gọi một thủ tục cục bộ.

Ví dụ ứng dụng điển hình:

- DCE RPC: Một hệ thống middleware thực sự được thiết kế để ẩn đi chi tiết về vị trí máy chủ, thiết lập giao tiếp, xử lý truyền thông điệp và chuyển đổi kiểu dữ liệu giữa client và server, ngay cả khi chúng chạy trên các kiến trúc phần cứng và hệ điều hành khác nhau.

- Java RMI: Trong Java, việc gọi một phương thức trên một đối tượng từ xa có thể trông giống hệt như gọi một phương thức trên đối tượng cục bộ trong mã nguồn.
- Transactional RPC: Các lời gọi thủ tục từ xa (RPC) có thể được đóng gói trong một giao dịch (transaction), đảm bảo tính nguyên tử (all-or-nothing) của một loạt các thao tác phân tán. Điều này thường được điều phối bởi một TP monitor.

5.2.3 Hệ thống kênh sự kiện / Publish-Subscribe (Event Channel / Publish-Subscribe Systems)

Mô hình này cho phép các thành phần giao tiếp một cách tách rời về tham chiếu và tách rời về thời gian.

Khái niệm:

- Publish-subscribe là một kiến trúc trong đó các nhà xuất bản (publisher) tạo ra các thông báo (notification) hoặc sự kiện (event), và các bên đăng ký (subscriber) nhận các thông báo này dựa trên sự quan tâm của họ.
- Giao tiếp diễn ra bằng cách mô tả các sự kiện mà một subscriber quan tâm, thay vì xác định rõ ràng người gửi và người nhận.
- Các thông báo thường được gắn thẻ bằng các thuộc tính (attributes). Subscriber cung cấp một mô tả sự kiện (ví dụ: các cặp thuộc tính-giá trị) để thể hiện sự quan tâm của mình (còn gọi là topic-based publish-subscribe hoặc content-based publish-subscribe nếu mô tả chi tiết hơn).
- Tách rời về thời gian: Người gửi không cần phải đang hoạt động khi người nhận nhận thông điệp, và ngược lại. Điều này được hỗ trợ bởi các hệ thống hàng đợi thông điệp (message-queuing systems).
- Tách rời về tham chiếu: Các bên giao tiếp không cần biết tên hoặc định danh của nhau.
- Shared Data Space: Một mô hình phối hợp khác, nơi các tiến trình giao tiếp hoàn toàn thông qua các "tuple" (bản ghi dữ liệu có cấu trúc) được đưa vào hoặc truy xuất từ một không gian dữ liệu dùng chung (tuple space). Các hoạt động out() (thêm tuple) và in() / rd() (lấy/đọc tuple) là các hoạt động blocking.

Ví dụ ứng dụng điển hình:

- TIB/Rendezvous: Một ví dụ về hệ thống publish/subscribe nơi các thông báo được gắn thẻ bằng các từ khóa phức hợp (ví dụ: news.comp.os.books) và subscriber đăng ký theo các từ khóa này.
- Hệ thống Microblog đơn giản: Người dùng đăng tin nhắn (tuple) vào một không gian dữ liệu dùng chung (ví dụ: <poster, topic, content>), và những người khác có thể đọc tin nhắn mà không cần biết ai đã đăng hoặc ai sẽ đọc.
- Message-Queuing Systems (MOM): Cung cấp khả năng lưu trữ thông điệp trung gian, cho phép người gửi và người nhận không cần phải hoạt động đồng thời. Ví dụ: IBM’s WebSphere message-queuing system (MQ).

- Message Brokers: Được sử dụng trong các hệ thống hàng đợi thông điệp để chuyển đổi định dạng hoặc ngữ nghĩa của thông điệp giữa các ứng dụng khác nhau, hoạt động như một gateway cấp ứng dụng.

5.2.4 Hệ thống dữ liệu tập trung (Data-centric Systems)

Trong ngữ cảnh hệ thống phân tán, "dữ liệu tập trung" thường ám chỉ các hệ thống nơi quản lý dữ liệu là mối quan tâm chính, và dữ liệu có thể được phân tán hoặc sao chép, nhưng vẫn duy trì một cái nhìn logic nhất quán về nó. Chương 1 đã giới thiệu "Hệ thống thông tin phân tán" là một lớp quan trọng.

Khái niệm:

- Hệ thống thông tin phân tán: Tập trung vào việc tích hợp các ứng dụng bằng cách cho phép chúng giao tiếp trực tiếp với nhau, thường xoay quanh ứng dụng cơ sở dữ liệu và thực hiện các hoạt động dưới dạng giao dịch.
- Các giao dịch tuân thủ thuộc tính ACID để đảm bảo tính toàn vẹn và nhất quán của dữ liệu.
- Giao dịch lồng nhau là cách tự nhiên để phân tán một giao dịch trên nhiều máy.
- Mặc dù dữ liệu có thể được phân tán vật lý hoặc sao chép, mục tiêu là cung cấp một cái nhìn nhất quán về dữ liệu, dù có thể chấp nhận các mức độ nhất quán yếu hơn như nhất quán liên tục hoặc nhất quán cuối cùng.

Ví dụ ứng dụng điển hình:

- Hệ thống xử lý giao dịch phân tán: Sử dụng TP monitor để phối hợp các giao dịch (có thể lồng nhau) trên các máy chủ/cơ sở dữ liệu khác nhau. Ví dụ, việc đặt chỗ ba chuyến bay khác nhau có thể được chia thành ba giao dịch con, mỗi giao dịch được quản lý độc lập.
- Các trang Web với nội dung động: Nội dung được lưu trữ trong một cơ sở dữ liệu ở phía máy chủ và được tạo ra động khi có yêu cầu từ client. Đây là một ví dụ về dữ liệu được quản lý tập trung (trong cơ sở dữ liệu máy chủ) nhưng được truy cập và hiển thị qua một hệ thống phân tán.
- Mạng lưới phân phối nội dung (Content Delivery Networks - CDN): Mặc dù mục tiêu là phân phối nội dung để tăng hiệu suất và khả dụng, các CDN quản lý tập trung việc sao chép và vị trí của các tài liệu Web (dữ liệu) trên nhiều máy chủ khác nhau để đưa chúng gần hơn với người dùng. Ví dụ như Akamai CDN điều hướng client đến máy chủ bản sao gần nhất/ít tải nhất để lấy tài liệu.
- Distributed Hash Tables (DHTs) như Chord: Mặc dù là hệ thống ngang hàng phân tán, chúng cung cấp một cách "tập trung logic" để tìm kiếm dữ liệu. Một khóa dữ liệu được ánh xạ đến một nút cụ thể chịu trách nhiệm duy trì dữ liệu đó, tạo ra một không gian định danh và dữ liệu có cấu trúc

5.3 Nêu vai trò của phần mềm trung gian (middleware) trong kiến trúc khách chủ phân tán, và liệt kê ba tính năng chính mà nó cung cấp.

Phần mềm trung gian (middleware) đóng vai trò trung tâm trong kiến trúc khách-chủ phân tán bằng cách hoạt động như một lớp phần mềm riêng biệt, được đặt một cách logic trên đầu các hệ điều hành tương ứng của các máy tính tạo nên hệ thống. Nó được thiết kế để cung cấp cho mỗi ứng dụng một giao diện nhất quán.

Về bản chất, phần mềm trung gian đối với hệ thống phân tán cũng giống như hệ điều hành đối với một máy tính: nó đóng vai trò quản lý tài nguyên, giúp các ứng dụng chia sẻ và triển khai hiệu quả các tài nguyên đó trên một mạng lưới. Mục tiêu chính của nó là ẩn giấu sự phân tán của các quy trình và tài nguyên trên nhiều máy tính. Sự "minh bạch phân tán" này có nghĩa là người dùng cuối và các ứng dụng không cần biết chính xác dữ liệu được lưu trữ ở đâu hoặc một phần công việc đang được xử lý trên máy tính nào.

Phần mềm trung gian cung cấp nhiều dịch vụ hữu ích cho nhiều ứng dụng, giúp tránh việc phải triển khai lại các chức năng chung một cách lặp đi lặp lại. Dưới đây là ba tính năng chính mà phần mềm trung gian cung cấp:

- Cơ sở vật chất để giao tiếp giữa các ứng dụng (Facilities for interapplication communication): Middleware cho phép các thành phần của một ứng dụng phân tán giao tiếp với nhau, cũng như cho phép các ứng dụng khác nhau trao đổi thông tin. Một ví dụ điển hình là Remote Procedure Call (RPC), cho phép một ứng dụng gọi một hàm được cài đặt và thực thi trên một máy tính từ xa như thể nó có sẵn tại chỗ.
- Dịch vụ bảo mật (Security services): Middleware tích hợp các tính năng bảo mật để giải quyết các mối đe dọa, chẳng hạn như đảm bảo rằng thông tin chỉ được tiết lộ cho các bên được ủy quyền và các thay đổi đối với dữ liệu chỉ được thực hiện một cách hợp lệ.
- Che giấu và phục hồi từ lỗi (Masking of and recovery from failures): Middleware giúp che giấu các lỗi trong hệ thống phân tán và hỗ trợ phục hồi từ chúng, làm cho hệ thống có thể tiếp tục cung cấp dịch vụ ngay cả khi có các thành phần bị lỗi. Đây là một khía cạnh quan trọng vì lỗi bộ phận là điều không thể tránh khỏi trong các hệ thống phân tán.

6 Câu 6

6.1 Phân loại ba loại dịch vụ trong SOA (cơ bản, tích hợp, quy trình) kèm ví dụ điển hình cho mỗi loại.

6.1.1 Dịch vụ cơ bản

Là những dịch vụ độc lập, thực hiện một chức năng kinh doanh cụ thể hoặc truy cập trực tiếp đến dữ liệu. Đây là các thành phần nhỏ nhất trong SOA, thường ánh xạ đến một chức năng hoặc thao tác trong một hệ thống hiện có.

Ví dụ: Dịch vụ CustomerInfoService cung cấp thông tin khách hàng dựa trên CustomerID. Nó có thể truy cập cơ sở dữ liệu để trả về thông tin như tên, địa chỉ, trạng thái tài khoản.

6.1.2 Dịch vụ tích hợp

Tổng hợp các dịch vụ cơ bản từ nhiều nguồn khác nhau (có thể từ các hệ thống không đồng nhất), xử lý logic trung gian, hoặc chuyển đổi định dạng dữ liệu giữa các dịch vụ. Dịch vụ tích hợp giúp tạo ra một giao diện thống nhất cho các dịch vụ cơ bản.

Ví dụ: Dịch vụ OrderProcessingService lấy thông tin từ CustomerInfoService, InventoryService và PaymentService để kiểm tra tình trạng hàng, xác thực thanh toán và cập nhật đơn hàng.

6.1.3 Dịch vụ quy trình

Dịch vụ quy trình mô hình hóa toàn bộ quy trình nghiệp vụ phức tạp, sử dụng các dịch vụ cơ bản và tích hợp như các bước trong quy trình. Thường được mô tả bằng ngôn ngữ BPM (Business Process Modeling), như BPEL (Business Process Execution Language).

Ví dụ: Dịch vụ LoanApprovalProcessService trong một ngân hàng thực hiện quy trình duyệt vay, bao gồm: kiểm tra điểm tín dụng, định giá tài sản đảm bảo, đánh giá rủi ro và cuối cùng phê duyệt khoản vay. Mỗi bước có thể là một dịch vụ con được phối hợp lại.

6.2 Trình bày vòng đời của một dịch vụ SOA, từ giai đoạn phát triển đến vận hành sản xuất, và những thách thức chính ở mỗi giai đoạn.

6.2.1 Giai đoạn phân tích và thiết kế dịch vụ

- Xác định nhu cầu nghiệp vụ và dịch vụ cần thiết.
- Thiết kế giao diện dịch vụ (service contract) – định nghĩa rõ input, output, giao thức (protocol), định dạng (XML, JSON...).
- Thách thức:
  - Hiểu sai hoặc không đầy đủ yêu cầu nghiệp vụ.
  - Thiết kế không thống nhất hoặc không tuân thủ nguyên tắc tái sử dụng.

- Thiếu chuẩn chung (naming, versioning, schema design…).

6.2.2 Giai đoạn phát triển

- Cài đặt logic nghiệp vụ theo thiết kế.
- Tuân thủ các chuẩn giao tiếp như SOAP, REST, gRPC.
- Viết unit test và mock test cho các thành phần dịch vụ.
- Thách thức:
  - Kết nối với hệ thống cũ (legacy systems).
  - Phụ thuộc vào dịch vụ bên thứ ba chưa sẵn sàng.
  - Khó kiểm thử vì tính phân tán và phi đồng bộ.

6.2.3 Giai đoạn kiểm thử

- Kiểm thử chức năng, hiệu năng, tích hợp.
- Kiểm thử bảo mật và độ ổn định.
- Thách thức:
  - Môi trường kiểm thử không phản ánh đúng môi trường thực tế.
  - Khó tái hiện lỗi khi có nhiều dịch vụ liên quan.
  - Đảm bảo backward compatibility (tương thích ngược).

6.2.4 Giai đoạn triển khai

- Đưa dịch vụ lên môi trường staging hoặc production.
- Sử dụng container (Docker), orchestrator (Kubernetes), CI/CD pipeline.
- Thách thức:
  - Triển khai không gián đoạn (zero downtime).
  - Phiên bản hóa dịch vụ (versioning) – tránh ảnh hưởng hệ thống khác.
  - Cấu hình bảo mật, certificate, load balancing…

6.2.5 Giai đoạn vận hành và giám sát

- Theo dõi tình trạng dịch vụ: latency, error rate, availability.
- Logging, tracing (dùng OpenTelemetry, ELK, Grafana...).
- Thách thức:
  - Khó phát hiện nguyên nhân lỗi trong môi trường phân tán.
  - Quản lý service discovery và traffic routing.
  - Đảm bảo SLA (Service Level Agreement) và khả năng mở rộng.

6.2.6 Giai đoạn bảo trì và phát triển tiếp theo

- Cập nhật tính năng, vá lỗi bảo mật.
- Ngưng hoạt động dịch vụ cũ, đảm bảo migration an toàn.
- Thách thức:
  - Tránh phá vỡ hợp đồng dịch vụ với client đang dùng.
  - Tái sử dụng logic cũ trong khi đảm bảo hiệu năng.
  - Quản lý vòng đời nhiều phiên bản song song.