Câu 1:

Hai đặc điểm quan trọng nhất của hệ thống phân tán là:

1. Tính đồng thời (Concurrency):

Trong hệ thống phân tán, nhiều tiến trình hoặc tác vụ có thể được thực thi đồng thời trên các nút khác nhau trong hệ thống. Điều này cho phép hệ thống xử lý nhiều yêu cầu cùng một lúc, tăng hiệu suất và khả năng đáp ứng. Ví dụ, một trang web thương mại điện tử có thể xử lý đồng thời nhiều đơn hàng từ nhiều người dùng khác nhau. Nếu không có tính đồng thời, hệ thống sẽ phải xử lý từng đơn hàng một, dẫn đến thời gian chờ đợi lâu và trải nghiệm người dùng kém.

2. Khả năng mở rộng (Scalability):

Hệ thống phân tán có thể tăng (hay giảm) quy mô linh hoạt theo nhu cầu. Cụ thể, khi khối lượng công việc hoặc số người dùng tăng lên, ta có thể bổ sung thêm các nút tính toán (thêm máy chủ, dịch vụ mới) vào hệ thống phân tán một cách dễ dàng, mà hệ thống vẫn hoạt động ổn định. Ví dụ, các dịch vụ như Netflix hay Google Clouds sử dụng hàng loạt máy chủ làm việc song song để phục vụ hàng triệu người truy cập cùng lúc – khi cần có thể thêm hay bỏ bớt máy chủ để điều chỉnh tải. Khả năng mở rộng giúp ứng dụng chịu được tải cao và phục vụ đông người dùng hơn so với ứng dụng đơn trên một máy duy nhất, đồng thời tận dụng tốt tài nguyên phần cứng (tăng hiệu suất và thông lượng tổng thể).

Ba lý do cơ bản khiến các ứng dụng phân tán phức tạp hơn so với các ứng dụng đơn lẻ, bao gồm:

1. Xử lý lỗi và độ tin cậy (fault tolerance):

Trong môi trường phân tán, bất kỳ nút nào hoặc đoạn mạng nào cũng có thể gặp sự cố độc lập – có thể một máy chủ bị sập, một kết nối mạng đứt, hoặc phần mềm của một dịch vụ bị lỗi. Khác với ứng dụng đơn lẻ (thường chỉ cần lo lỗi cục bộ), hệ phân tán phải thiết kế để chịu được các sự cố từng phần và tiếp tục hoạt động. Sự cố đơn lẻ và tính không xác định trong kết quả gọi hàm phân tán là hai vấn đề trọng yếu gây nhiều khó khăn nhất cho hệ thống phân tán. Ví dụ, trong một hệ thống chat như Zalo hay Facebook Messenger, nếu một máy chủ xử lý tin nhắn bị rớt mạng, toàn bộ dịch vụ không thể dừng lại mà phải có cơ chế chuyển sang máy khác hoặc gửi lại tin nhắn sau.

2. Tính đồng thời và phi xác định (concurrency and non-determinism):

Ứng dụng phân tán thường chạy song song trên nhiều nút, dẫn đến rất nhiều thao tác xảy ra đồng thời mà không có đồng hồ chung. Điều này gây ra vấn đề đồng bộ và nhất quán dữ liệu phức tạp. Việc nhiều tiến trình cùng truy

cập, cập nhật tài nguyên chia sẻ (ví dụ sổ cái giao dịch, hay mục tin nhắn) có thể gây ra xung đột (race condition) nếu không được xử lý đúng. Thêm vào đó, do dữ liệu và thao tác được lan truyền qua mạng với độ trễ khác nhau, kết quả thực thi có thể không xác định rõ thứ tự, gây khó khăn trong thiết kế. Thật vậy, độ đồng thời cao và tính mất định khi thông điệp truyền đi khiến concurrency được coi là một trong những thách thức phức tạp nhất của hệ thống phân tán. Ví dụ, trong một cơ sở dữ liệu phân tán, nếu hai máy chủ đồng thời ghi lên cùng một bản ghi, ta phải thiết kế cơ chế khóa hoặc kiểm tra phiên bản để tránh mất mát hay mâu thuẫn dữ liệu.

3. Giao tiếp qua mạng và độ trễ (networking and latency):

Các thành phần của hệ thống phân tán phải giao tiếp qua mạng, nên luôn tồn tại độ trễ truyền thông, khả năng thất lạc hoặc trễ gói tin. Không như chương trình đơn chạy cục bộ, mỗi lần gọi hàm, truy cập dữ liệu hay thông báo giữa các nút đều phải gửi qua mạng (thường là giao thức TCP/IP hoặc HTTP). Độ trễ này ảnh hưởng lớn đến hiệu năng và độ phức tạp của hệ thống – ví dụ, một yêu cầu phản hồi trong ứng dụng web phải trải qua nhiều bước gửi và nhận trên mạng, nên lập trình viên phải xử lý thêm cơ chế timeout, retry, hay circuit breaker. Bên cạnh đó, khi hệ thống phân tán quy mô lớn thì vấn đề đóng gói (marshalling) và giải mã dữ liệu (unmarshalling), cũng như thuật toán đồng bộ (như Paxos để đồng thuận) càng làm tăng độ khó thiết kế. Một ví dụ thực tế: dịch vụ phân tán như Google Maps hay Zalo phải xử lý hàng triệu yêu cầu vị trí và tin nhắn mỗi giây – tất cả giao tiếp qua internet với độ trễ không thể bằng 0, do đó hệ thống phải thiết kế smart cache, bản sao dữ liệu (replication) và cân bằng tải để đảm bảo phản hồi kịp thời.

Như vậy, so với ứng dụng đơn lẻ chỉ chạy trên một máy, ứng dụng phân tán phức tạp hơn nhiều vì phải xử lý nhiều vấn đề phân tán đồng thời: đảm bảo tính chịu lỗi khi các nút độc lập có thể hỏng, đồng bộ dữ liệu và lập trình song song không có bộ nhớ chung, và giao tiếp qua mạng với mọi giới hạn về băng thông, độ trễ hay mất gói. Các công ty công nghệ lớn như Google, Facebook, Zalo hay Netflix đã đầu tư sâu vào các thuật toán và hạ tầng phân tán để giải quyết những thách thức này, cho phép họ phục vụ quy mô hàng triệu người dùng hiệu quả mặc dù mô hình hệ thống rất phức tạp.

Câu 2:

Hiệu năng của hệ thống phân tán chịu ảnh hưởng bởi nhiều yếu tố phần cứng và mạng. Các yếu tố phần cứng bao gồm CPU (tốc độ xử lý), bộ nhớ (dung lượng và tốc độ truy cập), và kênh truyền (băng thông và độ trễ). Về mặt mạng,

băng thông và topology mạng đóng vai trò quan trọng trong việc quyết định tốc độ và khả năng mở rộng của hệ thống.

Phân tích các yếu tố:

1. Yếu tố phần cứng:

CPU: Tốc độ xử lý của CPU ảnh hưởng trực tiếp đến khả đáp ứng của hệ thống trong việc thực hiện các tác vụ tính toán. CPU càng nhanh, hệ thống càng có khả năng xử lý nhiều tác vụ đồng thời và giảm thời gian chờ đợi của người dùng.

Bộ nhớ: Dung lượng bộ nhớ lớn giúp hệ thống lưu trữ nhiều dữ liệu và ứng dụng, giảm thiểu việc truy cập vào bộ nhớ chậm (ổ cứng). Tốc độ truy cập bộ nhớ nhanh (RAM) giúp tăng tốc độ xử lý dữ liệu và cải thiện hiệu năng tổng thể. Việc thiếu bộ nhớ hoặc bộ nhớ chậm có thể gây ra tình trạng tắc nghẽn và làm chậm hệ thống.

Kênh truyền (Interconnect**):** Băng thông của kênh truyền quyết định tốc độ truyền dữ liệu giữa các thành phần của hệ thống phân tán. Độ trễ (latency) của kênh truyền ảnh hưởng đến thời gian phản hồi của hệ thống. Băng thông cao và độ trễ thấp là yếu tố quan trọng để đảm bảo hiệu năng tốt.

2. Yếu tố mạng:

Băng thông: Băng thông mạng là khả năng truyền tải dữ liệu trong một đơn vị thời gian. Mạng có băng thông lớn sẽ cho phép truyền tải nhiều dữ liệu hơn, giảm thời gian chờ đợi và cải thiện hiệu năng của hệ thống phân tán

Topology mạng: Topology mạng (cấu trúc liên kết) xác định cách các nút mạng (máy tính, máy chủ, thiết bị) kết nối với nhau. Các loại topology phổ biến bao gồm:

Topology hình sao (Star Topology): Tất cả các thiết bị đều kết nối với một thiết bị trung tâm (hub hoặc switch). Ưu điểm là dễ dàng quản lý và mở rộng, nhược điểm là nếu thiết bị trung tâm hỏng, toàn bộ mạng sẽ ngừng hoạt động.

Topology hình bus (Bus Topology): Tất cả các thiết bị đều kết nối vào một đường truyền chung. Ưu điểm là đơn giản, tiết kiệm chi phí, nhược điểm là khó mở rộng, khi một thiết bị hỏng có thể ảnh hưởng đến toàn bộ mạng.

Topology hình lưới (Mesh Topology): Mỗi thiết bị có thể kết nối trực tiếp với nhiều thiết bị khác. Ưu điểm là độ tin cậy cao, khả năng chịu lỗi tốt, nhược điểm là phức tạp và tốn kém.

Topology hình cây (Tree Topology): Kết hợp giữa topology hình sao và hình bus, có cấu trúc phân cấp. Ưu điểm là dễ dàng quản lý và mở rộng, nhược điểm là vẫn có điểm lỗi trung tâm.

Topology mạng hỗn hợp (Hybrid Topology): Kết hợp nhiều topology khác nhau. Ưu điểm là linh hoạt, có thể tận dụng ưu điểm của các topology khác, nhược điểm là phức tạp.

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

Đặc điểm	**Hệ điều hành phân tán (Distributed OS)**	**Hệ điều hành mạng (Network OS)**
Mục tiêu	Cung cấp một hệ thống thống nhất, minh bạch	Chia sẻ tài nguyên trong một mạng
Quản lý tài nguyên	Phức tạp, bao gồm chia sẻ, đồng bộ hóa, cân bằng tải, chịu lỗi, truy cập minh bạch	Đơn giản hơn, chủ yếu tập trung vào chia sẻ tài nguyên
Độ phức tạp	Cao hơn	Thấp hơn
Tính minh bạch	Cao	Thấp
Khả năng mở rộng	Rất cao	Hạn chế hơn

Tóm lại, hệ điều hành phân tán và hệ điều hành mạng có những yêu cầu khác nhau về quản lý tài nguyên do mục tiêu và kiến trúc khác nhau. Hệ điều hành phân tán tập trung vào việc tạo ra một hệ thống thống nhất, minh bạch, trong khi hệ điều hành mạng tập trung vào việc chia sẻ tài nguyên trong một mạng

<u>Câu 3</u>:

So sánh ba loại hệ thống phân tán: điện toán phân tán, thông tin phân tán và lan tỏa phân tán

Loại hệ thống	**Mục tiêu chính**	**Đặc điểm**	**Ví dụ**
Điện toán phân tán	Tăng hiệu suất tính toán	Nhiều máy tính phối hợp xử lý	HPC, xử lý song song
Thông tin phân tán	Chia sẻ dữ liệu	Dữ liệu lưu ở nhiều nơi	CSDL phân tán, web search
Lan tỏa phân tán	Môi trường phổ biến	Thiết bị nhỏ, nhúng, phân bố	IoT, cảm biến môi trường

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

Trong kiến trúc điện toán lưới (grid computing), có ba lớp chính: ứng dụng (application), middleware, và tài nguyên (resource). Mỗi lớp đóng một vai trò khác nhau, cùng nhau tạo nên một hệ thống phân tán mạnh mẽ và linh hoạt.

1. Lớp Ứng dụng (Application Layer):

Vai trò: Lớp này bao gồm các ứng dụng và dịch vụ được xây dựng trên nền tảng điện toán lưới. Đây là nơi người dùng cuối tương tác trực tiếp để thực hiện các tác vụ cụ thể, như phân tích dữ liệu, mô phỏng khoa học, hoặc xử lý hình ảnh.

Ví dụ: Ứng dụng khoa học tính toán, ứng dụng tài chính, ứng dụng y tế.

Đặc điểm: Các ứng dụng này thường yêu cầu tài nguyên tính toán lớn, bộ nhớ, và khả năng lưu trữ phân tán. Chúng được thiết kế để tận dụng sức mạnh của điện toán lưới thông qua việc sử dụng middleware để giao tiếp với các tài nguyên.

2. Lớp Middleware:

Vai trò: Middleware là lớp trung gian, cung cấp các dịch vụ và giao diện để các ứng dụng có thể truy cập và quản lý các tài nguyên trên lưới một cách dễ dàng và hiệu quả. Nó chịu trách nhiệm về việc tìm kiếm, ánh xạ, và phân phối công việc đến các tài nguyên phù hợp.

Ví dụ: Globus Toolkit, UNICORE, gLite, v.v.

Đặc điểm: Middleware giải quyết các vấn đề phức tạp như quản lý tài nguyên phân tán, điều phối công việc, xác thực và ủy quyền người dùng, quản lý dữ liệu, và xử lý lỗi. Nó giúp ứng dụng không cần quan tâm đến chi tiết kỹ thuật của việc truy cập các tài nguyên và tập trung vào logic nghiệp vụ.

3. Lớp Tài nguyên (Resource Layer):

Vai trò: Lớp này bao gồm các tài nguyên tính toán, lưu trữ, và các dịch vụ mạng có sẵn trên lưới. Nó cung cấp sức mạnh tính toán và khả năng lưu trữ mà các ứng dụng cần.

Ví dụ: Máy chủ, cụm máy chủ, hệ thống lưu trữ phân tán, cơ sở dữ liệu, các dịch vụ mạng.

Đặc điểm: Các tài nguyên này có thể phân tán trên nhiều địa điểm khác nhau, và middleware sẽ đảm bảo rằng chúng được sử dụng hiệu quả và phối hợp với nhau để đáp ứng yêu cầu của các ứng dụng.

<u>Câu 4</u>:

"Tính sẵn sàng" (availability) được xem là mục tiêu quan trọng nhất của hệ thống phân tán

Hệ thống phân tán thường được xây dựng để phục vụ nhiều người dùng đồng thời, trải rộng về mặt địa lý và thời gian.

Người dùng mong đợi hệ thống luôn sẵn sàng 24/7, đặc biệt là trong các dịch vụ quan trọng như ngân hàng, thương mại điện tử, y tế, điện toán đám mây.

Trong hệ thống phân tán, có nhiều thành phần (máy chủ, mạng, phần mềm, dịch vụ con...) có thể bị lỗi.

Hiệu năng tốt mà hệ thống không hoạt động thì không còn giá trị.

Tính mở, tính mở rộng, tính tương tác… chỉ hữu ích khi hệ thống vẫn đang hoạt động.

Vì thế, availability là nền tảng, là mục tiêu quan trọng nhất để đảm bảo hệ thống phân tán phát huy hiệu quả. Trong hệ thống phân tán, nơi mọi thành phần đều có thể gặp lỗi, "tính sẵn sàng" là yếu tố then chốt để hệ thống duy trì hoạt động ổn định, liên tục và đáng tin cậy cho người dùng**.** Vì vậy, nó được xem là mục tiêu quan trọng nhất trong thiết kế và vận hành hệ thống phân tán.

Ví dụ thực tế: Facebook Messenger đảm bảo bạn luôn gửi/nhận được tin nhắn, dù có thể tin hiển thị chưa "đồng bộ" hoàn toàn giữa các thiết bị – tức vẫn available, nhưng dữ liệu có thể chưa "consistent" tức thì.

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.

Tính trong suốt trong hệ thống phân tán đề cập đến việc che giấu sự phức tạp của hệ thống khỏi người dùng hoặc nhà phát triển, khiến họ cảm thấy như đang sử dụng một hệ thống đơn lẻ. Ba hình thức chính là:

a. Trong suốt về truy nhập (Access Transparency):

- **Định nghĩa**: Người dùng hoặc ứng dụng có thể truy cập dữ liệu hoặc tài nguyên mà không cần biết chúng được lưu trữ hoặc xử lý ở đâu, hoặc sử dụng giao thức nào.
- **Ví dụ**: Một ứng dụng email như Gmail cho phép người dùng đọc email từ trình duyệt hoặc ứng dụng di động mà không cần biết dữ liệu được lưu trên máy chủ nào hay giao thức (IMAP, POP3) được sử dụng.
- **So sánh**: Tính trong suốt này tập trung vào việc cung cấp giao diện thống nhất cho các tài nguyên khác nhau, giúp người dùng không cần hiểu về các chi tiết kỹ thuật.

b. Trong suốt về vị trí (Location Transparency):

- **Định nghĩa**: Người dùng hoặc ứng dụng không cần biết vị trí vật lý của tài nguyên (ví dụ: máy chủ, cơ sở dữ liệu) để truy cập chúng.
- **Ví dụ**: Khi bạn truy cập một trang web như Wikipedia, bạn không cần biết trang web được lưu trữ trên máy chủ ở Mỹ, châu Âu hay châu Á. Hệ thống tự động định tuyến yêu cầu đến máy chủ gần nhất.

- **So sánh**: Khác với trong suốt về truy nhập, tính trong suốt về vị trí che giấu chi tiết địa lý hoặc mạng của tài nguyên, tập trung vào việc ẩn đi thông tin về nơi tài nguyên được đặt.

c. Trong suốt về lỗi (Failure Transparency):

- **Định nghĩa**: Hệ thống che giấu các lỗi (như hỏng máy chủ, mất kết nối) khỏi người dùng, đảm bảo dịch vụ vẫn hoạt động bình thường.
- **Ví dụ**: Trong hệ thống lưu trữ đám mây như Google Drive, nếu một máy chủ lưu trữ tệp bị lỗi, hệ thống sẽ tự động chuyển sang máy chủ khác mà người dùng không nhận thấy gián đoạn khi tải tệp.
- **So sánh**: Tính trong suốt về lỗi tập trung vào việc duy trì hoạt động liên tục bất chấp sự cố, trong khi hai loại trên liên quan đến cách tài nguyên được truy cập hoặc định vị.

So sánh tổng quát:

- **Mục tiêu chung**: Cả ba loại trong suốt đều nhằm đơn giản hóa trải nghiệm người dùng và giảm độ phức tạp khi tương tác với hệ thống phân tán.
- **Khác biệt**:
  - **Truy nhập**: Ẩn chi tiết giao thức hoặc cách truy cập tài nguyên.
  - **Vị trí**: Ẩn thông tin địa lý hoặc mạng của tài nguyên.
  - **Lỗi**: Ẩn các sự cố để đảm bảo hoạt động liên tục.
- **Độ phức tạp triển khai**: Trong suốt về lỗi thường khó triển khai nhất vì đòi hỏi cơ chế dự phòng (redundancy) và phục hồi (recovery) phức tạp.

<u>Câu 5</u>:

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

Kiến trúc phân cấp, với các nút có vai trò khác nhau (ví dụ: một hoặc một vài nút trung tâm điều khiển các nút khác), thường có lợi thế về hiệu suất và khả năng quản lý, nhưng có thể kém linh hoạt và dễ bị tổn thương hơn khi nút trung tâm gặp sự cố. Ngược lại, kiến trúc ngang hàng, với tất cả các nút đều có vai trò tương đương, thường linh hoạt hơn, có khả năng mở rộng tốt hơn và ít bị ảnh hưởng bởi sự cố của một nút, nhưng có thể kém hiệu quả hơn trong một số trường hợp và khó quản lý hơn

Kiến trúc phân cấp:

<u>Ưu điểm</u>:

Hiệu suất cao: Các nút trung tâm có thể tối ưu hóa việc xử lý và phân phối công việc, giúp tăng tốc độ xử lý tổng thể của hệ thống.

Quản lý tập trung: Dễ dàng quản lý, giám sát và bảo trì hệ thống do có một hoặc một vài nút trung tâm.

Giảm tải mạng: Các nút trung tâm có thể giảm tải mạng bằng cách quản lý các yêu cầu và phân phối dữ liệu một cách hiệu quả.

<u>Nhược điểm</u>:

Điểm lỗi đơn: Nếu một nút trung tâm gặp sự cố, toàn bộ hệ thống có thể bị ảnh hưởng hoặc ngừng hoạt động.

Khả năng mở rộng hạn chế: Việc mở rộng hệ thống có thể khó khăn và tốn kém do cần phải nâng cấp hoặc thay thế các nút trung tâm.

Độ trễ có thể cao hơn: Trong một số trường hợp, việc chuyển tiếp dữ liệu qua các nút trung tâm có thể làm tăng độ trễ, đặc biệt khi khoảng cách giữa các nút lớn.

Kiến trúc ngang hàng:

Ưu điểm:

Linh hoạt và khả năng mở rộng tốt: Dễ dàng thêm hoặc bớt các nút mà không ảnh hưởng đến toàn bộ hệ thống.

Ít điểm lỗi đơn: Sự cố của một nút không ảnh hưởng đến toàn bộ hệ thống.

Độ trễ thấp: Dữ liệu có thể được truyền trực tiếp giữa các nút, giảm độ trễ.

Nhược điểm:

Quản lý phức tạp: Khó quản lý, giám sát và bảo trì hệ thống do không có nút trung tâm.

Hiệu suất có thể thấp hơn: Trong một số trường hợp, việc phân phối công việc giữa các nút ngang hàng có thể kém hiệu quả hơn so với kiến trúc phân cấp.

Bảo mật: Có thể khó đảm bảo an ninh và bảo mật do tất cả các nút đều có vai trò tương đương.

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

1. Mô hình phân tầng (Layered Architecture):

Định nghĩa: Hệ thống được chia thành các lớp chồng lên nhau, mỗi lớp thực hiện một tập hợp các chức năng cụ thể và giao tiếp với các lớp liền kề.

Cách thức hoạt động: Dữ liệu và yêu cầu xử lý đi qua các lớp theo một thứ tự xác định. Lớp trên nhận dữ liệu từ lớp dưới, xử lý và chuyển kết quả lên lớp trên.

Ví dụ ứng dụng:

Mô hình OSI cho mạng máy tính: Các lớp như lớp vật lý, lớp liên kết dữ liệu, lớp mạng, lớp truyền tải, lớp phiên, lớp trình bày, và lớp ứng dụng, mỗi lớp có chức năng riêng biệt và giao tiếp với nhau để truyền tải dữ liệu.

Hệ thống quản lý cơ sở dữ liệu: Các lớp có thể bao gồm lớp truy cập dữ liệu, lớp xử lý truy vấn, và lớp lưu trữ dữ liệu.

2. Mô hình đối tượng phân tán (Distributed Object Model):

Định nghĩa: Các đối tượng được phân tán trên các máy tính khác nhau, nhưng vẫn có thể tương tác với nhau như thể chúng là một phần của một hệ thống duy nhất.

Cách thức hoạt động: Các đối tượng giao tiếp thông qua các giao diện (interface) đã được định nghĩa. Khi một đối tượng cần sử dụng một đối tượng khác, nó sẽ gọi phương thức trên giao diện của đối tượng đó.

Ví dụ ứng dụng:

CORBA (Common Object Request Broker Architecture): Một kiến trúc cho phép các đối tượng trên các hệ thống khác nhau giao tiếp với nhau.

Dịch vụ web (Web services): Các ứng dụng có thể truy cập các dịch vụ được cung cấp bởi các máy chủ khác nhau thông qua các giao thức như SOAP hoặc REST.

3. Mô hình kênh sự kiện (Event-Driven Architecture):

Định nghĩa: Hệ thống phản ứng với các sự kiện được phát ra từ các thành phần khác nhau.

Cách thức hoạt động: Các thành phần trong hệ thống đăng ký lắng nghe các sự kiện cụ thể. Khi một sự kiện xảy ra, nó sẽ được phát đi và các thành phần đã đăng ký sẽ nhận được thông báo và thực hiện các hành động tương ứng.

Ví dụ ứng dụng:

Hệ thống giao dịch chứng khoán: Các sự kiện như giá cổ phiếu thay đổi, lệnh mua/bán được thực hiện sẽ kích hoạt các hành động như cập nhật giao dịch, thông báo cho nhà đầu tư.

Hệ thống giám sát và cảnh báo: Các cảm biến gửi dữ liệu về hệ thống, các sự kiện như nhiệt độ vượt quá ngưỡng, máy móc hỏng hóc sẽ kích hoạt cảnh báo.

4. Mô hình dữ liệu tập trung (Shared Data Model):

Định nghĩa: Dữ liệu được lưu trữ tập trung và các thành phần trong hệ thống truy cập dữ liệu này.

Cách thức hoạt động: Các thành phần trong hệ thống có thể đọc và ghi dữ liệu vào một kho dữ liệu chung.

Ví dụ ứng dụng:

Hệ thống quản lý quan hệ khách hàng (CRM): Dữ liệu khách hàng, lịch sử giao dịch, thông tin liên lạc được lưu trữ tập trung.

Hệ thống quản lý nội dung (CMS): Dữ liệu bài viết, hình ảnh, video được lưu trữ tập trung và các trang web có thể truy cập để hiển thị nội dung.

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

Trong kiến trúc client-server phân tán, phần mềm trung gian (middleware) đóng vai trò là cầu nối giữa các ứng dụng khác nhau, giúp chúng giao tiếp và chia sẻ dữ liệu một cách hiệu quả. Ba tính năng chính mà middleware cung cấp là: giao tiếp, tích hợp và quản lý.

Giao tiếp: Middleware cung cấp các giao thức và cơ chế để các ứng dụng có thể trao đổi thông tin, bất kể chúng nằm trên cùng một máy hay trên các máy khác nhau, thậm chí khác hệ điều hành. Điều này giúp các ứng dụng phân tán có thể hoạt động như một hệ thống thống nhất.

Tích hợp: Middleware giúp kết nối các ứng dụng khác nhau, kể cả những ứng dụng được phát triển bởi các nhà cung cấp khác nhau, với các công nghệ và cơ sở dữ liệu khác nhau. Nó cho phép các ứng dụng này chia sẻ dữ liệu và chức năng, tạo ra một dịch vụ thống nhất cho người dùng.

Quản lý: Middleware cung cấp các dịch vụ quản lý như cân bằng tải, bảo mật, và xử lý lỗi, giúp các ứng dụng phân tán hoạt động ổn định và hiệu quả hơn. Nó cũng có thể giúp đơn giản hóa việc phát triển các ứng dụng phân tán bằng cách cung cấp các công cụ và dịch vụ hỗ trợ.

Ví dụ, một ứng dụng frontend trên Windows có thể giao tiếp và lấy dữ liệu từ một máy chủ backend trên Linux thông qua middleware, mà người dùng cuối không hề nhận ra sự khác biệt.

Câu 6:

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

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

Vòng đời của một dịch vụ SOA, từ phát triển đến vận hành, bao gồm các giai đoạn chính: phát triển, kiểm thử, triển khai, và bảo trì/nâng cấp. Mỗi giai đoạn đều có những thách thức riêng.

Giai đoạn phát triển:

Xác định dịch vụ**:** Thách thức lớn nhất là xác định các dịch vụ phù hợp với kiến trúc SOA, đảm bảo chúng độc lập, có thể tái sử dụng và cung cấp các chức năng kinh doanh rõ ràng.

Thiết kế giao diện dịch vụ**:** Việc thiết kế giao diện dịch vụ (contract) cần phải chính xác, rõ ràng, và tuân thủ các chuẩn chung để đảm bảo khả năng tương tác giữa các dịch vụ và hệ thống khác.

Lựa chọn công nghệ**:** Việc lựa chọn công nghệ phù hợp cho việc phát triển dịch vụ, như ngôn ngữ lập trình, framework, và nền tảng, cũng là một thách thức.

Quản lý version**:** Các dịch vụ SOA cần được quản lý version một cách hiệu quả để tránh xung đột và đảm bảo tính tương thích khi cập nhật.

Giai đoạn kiểm thử:

Kiểm thử đơn vị: Kiểm thử đơn vị các dịch vụ cần được thực hiện kỹ lưỡng để đảm bảo tính đúng đắn và hiệu suất của từng dịch vụ.

Kiểm thử tích hợp: Kiểm thử tích hợp giữa các dịch vụ cần được thực hiện để đảm bảo các dịch vụ hoạt động đúng khi kết hợp với nhau.

Kiểm thử hiệu năng: Kiểm thử hiệu năng cần được thực hiện để đảm bảo các dịch vụ đáp ứng được yêu cầu về hiệu suất và khả năng mở rộng.

Kiểm thử bảo mật: Kiểm thử bảo mật cần được thực hiện để đảm bảo các dịch vụ an toàn và không bị tấn công.

Giai đoạn triển khai:

Triển khai dịch vụ**:** Việc triển khai dịch vụ cần phải được thực hiện một cách có kế hoạch và có thể tự động hóa để giảm thiểu rủi ro.

Quản lý môi trường**:** Cần có môi trường triển khai phù hợp, bao gồm cả môi trường phát triển, kiểm thử, và sản xuất.

Giám sát**:** Cần có hệ thống giám sát để theo dõi tình trạng hoạt động của các dịch vụ và phát hiện sớm các vấn đề.

Giai đoạn bảo trì và nâng cấp:

Bảo trì dịch vụ**:** Việc bảo trì các dịch vụ cần được thực hiện thường xuyên để đảm bảo chúng hoạt động ổn định và không bị lỗi thời.

Nâng cấp dịch vụ**:** Việc nâng cấp dịch vụ cần được thực hiện một cách cẩn thận để đảm bảo tính tương thích với các hệ thống khác và không làm gián đoạn hoạt động của hệ thống.

Quản lý phiên bản**:** Cần có quy trình quản lý phiên bản dịch vụ để theo dõi các thay đổi và đảm bảo tính nhất quán.

Cập nhật tài liệu**:** Cần cập nhật tài liệu về dịch vụ để đảm bảo rằng các nhà phát triển và người dùng cuối có thể sử dụng dịch vụ một cách hiệu quả.