Nhóm 6
Lớp: Hệ thống thông tin
Họ và tên:

1. Vũ Hồng Ngọc - B24CHHT087
2. Nguyễn Thị Hảo - B24CHHT069
3. Nguyễn Phương Đông - B24CHHT062

Câu 1:

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

Hai đặc điểm quan trọng nhất của hệ thống phân tán

1. Tập hợp các thành phần tính toán độc lập (Autonomous Nodes)

- Mỗi node là một thực thể riêng biệt, có thể là phần cứng (máy tính) hoặc phần mềm (tiến trình).
- Không có đồng hồ chung giữa các node → dễ dẫn đến vấn đề đồng bộ hóa và phối hợp hành động.
- Mỗi node có thể hoạt động độc lập, nhưng để hình thành hệ thống, chúng phải hợp tác.

2. Hoạt động như một hệ thống thống nhất (Single Coherent System)

- Người dùng/ứng dụng nhìn thấy toàn hệ thống như một thể thống nhất, không phân biệt vị trí, nơi lưu trữ hay việc sao lưu dữ liệu.
- Đây là mục tiêu của tính minh bạch phân tán (distribution transparency).
- Tuy nhiên, việc ẩn hoàn toàn các lỗi và độ trễ là không thể – đặc biệt khi một phần của hệ thống bị lỗi (partial failure).

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?

Ba lý do cơ bản khiến ứng dụng phân tán phức tạp hơn ứng dụng đơn lẻ

1. Môi trường mạng không ổn định

- Mạng có thể mất gói, chậm trễ, hoặc không đáng tin cậy.
- Không giống như ứng dụng đơn lẻ, nơi các thành phần đều trên một máy và giao tiếp nội bộ, hệ phân tán phải xử lý độ trễ và lỗi kết nối.

2. Không có bộ nhớ chung & đồng hồ chung

- Các node không thể chia sẻ trực tiếp trạng thái hoặc dữ liệu như trên một máy đơn.
- Đồng bộ hóa thời gian và dữ liệu giữa các node là vấn đề khó 1han, gây phức tạp cho việc đồng bộ hành vi và thứ tự thực thi.

3. Xử lý lỗi khó 1han hơn

- Trong hệ phân tán, lỗi từng phần (partial failures) là phổ biến: một node hoặc kết nối có thể chết, nhưng phần còn lại vẫn hoạt động.

- Phát hiện và xử lý lỗi không đơn giản như dừng chương trình trên một máy đơn – vì không thể phân biệt giữa "chậm" và "chết".

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?

1. Yếu tố phần cứng

a. CPU (bộ xử lý)

- Tốc độ xử lý của từng node quyết định khả năng xử lý dữ liệu cục bộ.
- Sự không đồng đều giữa các CPU trong hệ thống có thể gây mất cân bằng tải, ảnh hưởng đến tổng thể hiệu năng.

b. Bộ nhớ (RAM)

- Bộ nhớ hạn chế gây thắt cổ chai trong xử lý, đặc biệt nếu hệ thống cần lưu trữ trạng thái tạm thời hoặc cache dữ liệu.
- Sự không đồng nhất về dung lượng và tốc độ bộ nhớ giữa các node ảnh hưởng đến khả năng phối hợp.

c. Kênh truyền nội bộ (Bus, I/O)

- Ảnh hưởng đến tốc độ đọc/ghi dữ liệu giữa CPU và bộ nhớ hoặc đĩa cứng.
- Nếu bus nội bộ chậm, thì node đó trở thành điểm nghẽn, làm giảm hiệu năng toàn hệ thống.

2. Yếu tố mạng

a. Băng thông (Bandwidth)

- Quyết định khả năng truyền dữ liệu giữa các node.
- Băng thông thấp sẽ gây ra tắc nghẽn khi truyền thông tin, đặc biệt với các ứng dụng yêu cầu truyền lượng lớn dữ liệu (video streaming, sao lưu…).

b. Topology (cấu trúc mạng)

- Topology mạng (như hình sao, vòng, lưới, hoặc mạng phân cấp) ảnh hưởng đến:
  - Tốc độ truy cập dữ liệu
  - Độ trễ giữa các node
  - Độ bền khi có node bị lỗi (redundancy)
- Ví dụ: mạng hình sao có thể nhanh nhưng dễ bị lỗi toàn bộ khi node trung tâm hỏng.

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?

Do các nguyên nhân sau đây:

1. Network Operating System (NOS)

- Các máy tính hoạt động độc lập, có hệ điều hành riêng.
- Mỗi máy tự quản lý tài nguyên riêng của mình.

- Người dùng chủ động điều khiển tài nguyên từ xa (thông qua FTP, SSH, v.v).
- Mạng chủ yếu dùng để chia sẻ tài nguyên, không điều phối tập trung.

Quản lý tài nguyên rời rạc, ít liên kết chặt chẽ.

2. Distributed Operating System (DOS)

- Các máy được quản lý như một hệ thống duy nhất.
- Người dùng không cần biết tài nguyên ở đâu – hệ thống phân bổ tài nguyên tự động.
- Tài nguyên được ảo hóa và phân chia động, ví dụ: tiến trình A trên node 1 dùng RAM từ node 2.

Quản lý tài nguyên tập trung và minh bạch, đòi hỏi phức tạp hơn:

- Theo dõi trạng thái tài nguyên toàn hệ thống
- Cân bằng tải, di chuyển tiến trình
- Xử lý lỗi phần tử (node chết, mạng gián đoạn...)

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?

Nêu và so sánh ba loại hệ thống phân tán:

a. Điện toán phân tán hiệu năng cao (High Performance Distributed Computing - HPC)

- Mục tiêu: Tăng khả năng xử lý và hiệu năng bằng cách kết hợp nhiều máy tính để thực hiện tác vụ lớn.
- Ví dụ:
    - Cluster computing: cụm máy tính kết nối LAN, phần cứng tương tự nhau.
    - Grid computing: các máy tính không đồng nhất, phân bố nhiều tổ chức, kết nối qua Internet.
    - Cloud computing: cấp phát tài nguyên động theo yêu cầu (IaaS, PaaS, SaaS).

b. Hệ thống thông tin phân tán (Distributed Information Systems)

- Mục tiêu: Trao đổi và xử lý dữ liệu giữa các hệ thống phần mềm khác nhau trên mạng.
- Ví dụ:
    - Hệ thống giao dịch ngân hàng, thương mại điện tử, hệ thống đặt vé máy bay...
- Công nghệ: RPC, MOM, cơ sở dữ liệu phân tán, EAI (Enterprise Application Integration), TP Monitor...

c. Hệ thống lan tỏa phân tán (Pervasive/Ubiquitous Distributed Systems)

- Mục tiêu: Tích hợp công nghệ vào môi trường sống, tạo sự tương tác tự nhiên giữa hệ thống và con người.
- Ví dụ:
    - Thiết bị IoT, cảm biến môi trường, thiết bị di động...

- Đặc điểm:
    - Tự động, nhận biết ngữ cảnh, tiêu thụ năng lượng thấp, thường dùng mạng cảm biến không dây.
    - Tập trung vào tính ẩn mình (invisible) và mượt mà (seamless) trong tương tác.

Bảng so sánh

Tiêu chí	HPC	Thông tin phân tán	Lan tỏa phân tán
Mục tiêu chính	Tăng hiệu năng xử lý	Chia sẻ, tích hợp thông tin	Tương tác tự nhiên, tự động
Kiến trúc	Cluster, grid, cloud	CSDL phân tán, middleware	Sensor, mobile, IoT
Đặc trưng kỹ thuật	Sức mạnh tính toán	Quản lý giao dịch, tích hợp	Nhận biết ngữ cảnh, tiết kiệm

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

Kiến trúc điện toán lưới (Grid Computing Architecture)
Hệ thống điện toán lưới gồm 5 lớp chính, trong đó 3 lớp quan trọng thường được nhấn mạnh:

a. Fabric Layer (Lớp hạ tầng tài nguyên)

- Chức năng: Truy cập và quản lý tài nguyên vật lý như CPU, bộ nhớ, lưu trữ...
- Ví dụ: Interface để kiểm tra trạng thái máy, khóa tài nguyên, truy xuất dữ liệu từ máy chủ.

Vai trò: Cung cấp API truy xuất tài nguyên cục bộ trên từng node của lưới.

b. Resource Layer (Lớp tài nguyên)

- Chức năng: Quản lý từng tài nguyên riêng lẻ, như khởi tạo tiến trình, cấp phát bộ nhớ, thực hiện bảo mật.
- Ví dụ: Dịch vụ xác thực người dùng khi truy cập một máy tính cụ thể.

Vai trò: Quản lý truy cập và bảo vệ từng đơn vị tài nguyên.

c. Middleware / Collective Layer (Lớp tập thể)

- Chức năng: Điều phối nhiều tài nguyên cùng lúc → thực hiện lập lịch, tìm kiếm, sao lưu.
- Ví dụ: Tìm tài nguyên trống, lập lịch xử lý công việc, quản lý bản sao dữ liệu.

Vai trò: Tạo nên khả năng phối hợp giữa nhiều node trong môi trường không đồng nhất.

d. Connectivity Layer

- Chức năng: Giao tiếp giữa các node, cung cấp giao thức truyền thông, xác thực, ví dụ: TCP/IP, TLS.
- Vai trò: Đảm bảo dữ liệu truyền an toàn, đúng đích.

e. Application Layer

- Chức năng: Chứa ứng dụng thực tế chạy trên grid, ví dụ: mô phỏng thời tiết, phân tích gene.
- Vai trò: Là nơi người dùng tương tác với lưới, sử dụng tài nguyên thông qua các API.

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?

1. Định nghĩa "Tính sẵn sàng"

Tính sẵn sàng (Availability) là khả năng hệ thống đáp ứng yêu cầu của người dùng tại mọi thời điểm, ngay cả khi có một phần hệ thống bị lỗi.

2. Vì sao availability là mục tiêu hàng đầu?

a. Hệ thống phân tán dễ gặp lỗi cục bộ

- Trong hệ phân tán, có nhiều node hoạt động độc lập → xác suất một thành phần bị lỗi luôn tồn tại.
- Nếu không thiết kế tốt, lỗi một node có thể làm gián đoạn cả hệ thống.
- Vì vậy, phải đảm bảo rằng một phần lỗi không làm gián đoạn toàn bộ hệ thống → tính sẵn sàng là ưu tiên hàng đầu.

b. Tính sẵn sàng ảnh hưởng trực tiếp đến trải nghiệm người dùng

- Người dùng không quan tâm hệ thống "phân tán" hay "tập trung" – họ chỉ quan tâm liệu dịch vụ có hoạt động hay không.
- Hệ thống ngừng phản hồi = mất lòng tin, mất khách hàng, mất doanh thu.

c. Yêu cầu thiết yếu trong các ứng dụng quan trọng

- Ví dụ: ngân hàng trực tuyến, đặt vé, dịch vụ y tế, nền tảng học tập, video streaming…
- Nếu các dịch vụ này không khả dụng 24/7, hậu quả có thể nghiêm trọng (tài chính, pháp lý, xã hội...).

d. Tính sẵn sàng là tiền đề cho các mục tiêu khác

- Hiệu năng, bảo mật, minh bạch sẽ trở nên vô nghĩa nếu hệ thống không sẵn sàng.
- Một hệ thống "an toàn nhưng không thể truy cập được" là hệ thống vô dụng.

3. Để đạt được tính sẵn sàng, hệ phân tán phải làm gì?

- Sao lưu (replication): giữ nhiều bản sao để dự phòng.

- Phát hiện và xử lý lỗi cục bộ (fault detection & recovery).
- Thiết kế không có điểm chết (no single point of failure).
- Chuyển hướng truy cập khi có sự cố (failover).

Hệ thống phân tán có nhiều node, dễ xảy ra lỗi cục bộ. Nếu không đảm bảo tính sẵn sàng, người dùng sẽ không thể sử dụng hệ thống, làm mất đi ý nghĩa cốt lõi của phân tán. Do đó, tính sẵn sàng là mục tiêu quan trọng nhất, và là nền tảng cho mọi tính năng khác.

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?

"Tính trong suốt" (Transparency) giúp ẩn đi sự phức tạp của hệ thống phân tán đối với người dùng và lập trình viên.

1. Tính trong suốt về truy nhập (Access Transparency)

Định nghĩa:

Người dùng hoặc ứng dụng không cần quan tâm đến cách truy cập tài nguyên – dù là cục bộ hay từ xa, cách truy cập vẫn giống nhau.

Ví dụ:

- Truy cập file trên máy cục bộ và file trên máy chủ qua NFS đều thông qua lệnh open("file.txt").
- Lập trình viên không cần biết file nằm trên đâu – truy cập giống hệt.

Mục tiêu:

Ẩn đi sự khác biệt về phương thức truy cập (gọi hàm, gửi thông điệp, truyền file, v.v.).

2. Tính trong suốt về vị trí (Location Transparency)

Định nghĩa:

Người dùng không cần biết tài nguyên đang nằm ở đâu trong hệ thống.

Ví dụ:

- Bạn nhập URL https://example.com/image.jpg mà không biết file ảnh nằm ở server nào – hệ thống tự định tuyến đến nơi chứa ảnh.
- Tên miền ẩn đi vị trí IP thật của server, giúp truy cập trở nên "vô danh về vị trí".

Mục tiêu:

Cho phép di chuyển tài nguyên dễ dàng mà không cần thay đổi cách truy cập của người dùng.

3. Tính trong suốt về lỗi (Failure Transparency)

Định nghĩa:

Hệ thống vẫn tiếp tục hoạt động ngay cả khi một phần của nó bị lỗi, ẩn lỗi khỏi người dùng nếu có thể.

Ví dụ:

- Khi một node máy chủ bị lỗi, hệ thống tự động chuyển hướng truy cập sang bản sao (replica) – người dùng không bị gián đoạn.

- Ví dụ: dịch vụ video streaming tự động chuyển sang server dự phòng nếu server chính hỏng.

Mục tiêu:
Ẩn đi lỗi phần tử trong hệ thống phân tán và đảm bảo tính sẵn sàng.

So sánh ba hình thức:

Tiêu chí	Truy nhập	Vị trí	Lỗi
Ẩn điều gì?	Cách truy cập tài nguyên	Vị trí vật lý của tài nguyên	Sự cố/lỗi phần tử trong hệ thống
Lợi ích chính	Giao diện truy cập thống nhất	Dễ di chuyển tài nguyên	Tăng độ tin cậy và sẵn sàng
Ví dụ	open("file.txt") cho cả file từ xa	URL truy cập dữ liệu ẩn IP server	Tự động chuyển sang bản sao nếu lỗi
Thách thức	Đảm bảo API thống nhất	Cập nhật bảng định vị khi tài nguyên di chuyển	Không thể ẩn mọi lỗi hoàn toàn (e.g., mất kết nối toàn bộ)

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

1. Tính mở (Openness) trong hệ thống phân tán là gì?

Tính mở là khả năng của một hệ thống cho phép kết nối, mở rộng, tương tác và tích hợp với các hệ thống khác, bất kể nền tảng phần cứng hay phần mềm.
Tính mở được thể hiện qua:

- Chuẩn hóa giao diện (interface): dùng các giao thức chung (HTTP, REST, gRPC…).
- Khả năng mở rộng: dễ dàng thêm dịch vụ, node mới vào hệ thống.
- Hỗ trợ di động phần mềm: ứng dụng có thể chạy trên nhiều nền tảng khác nhau.

2. Khả năng tương tác (Interoperability) là gì?

Khả năng tương tác là khả năng các thành phần từ các hệ thống khác nhau có thể làm việc cùng nhau một cách trơn tru.
Ví dụ:

- Một ứng dụng .NET có thể giao tiếp với dịch vụ web viết bằng Java thông qua SOAP/REST.
- Một hệ thống thanh toán sử dụng API từ ngân hàng bên ngoài.

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. |

Cụ thể:

Tính mở cung cấp…	→ Giúp hệ thống có thể…
Các giao thức chuẩn (standard protocols)	Giao tiếp dễ dàng với hệ thống bên ngoài
Giao diện rõ ràng, định nghĩa tốt	Cho phép các ứng dụng khác gọi và sử dụng
Khả năng mở rộng linh hoạt	Thêm/bỏ thành phần mà không phá vỡ toàn hệ thống
Mô hình chính sách – cơ chế (policy vs mechanism)	Điều chỉnh được quy tắc tương tác với bên ngoài

Tóm lại:

- Tính mở là kiến trúc, khả năng tương tác là kết quả.
- Một hệ thống muốn tương tác tốt với hệ thống khác, phải được thiết kế mở từ đầu: chuẩn hóa, linh hoạt và mở rộng được.

Câu 5

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

1. Kiến trúc phân cấp (Hierarchical Architecture)

Ưu điểm:

- Dễ quản lý: có cấu trúc rõ ràng (thường dạng cây), dễ giám sát và điều khiển.
- Phân quyền rõ ràng: các tầng trên có thể kiểm soát các tầng dưới.
- Tối ưu cho tìm kiếm và điều phối: sử dụng các node trung tâm để điều phối và định tuyến nhanh.

Nhược điểm:

- Điểm nghẽn và điểm lỗi trung tâm: nếu một node trung tâm (ví dụ: server) bị lỗi, toàn bộ hệ thống hoặc nhánh có thể bị ảnh hưởng.
- Không linh hoạt: khó mở rộng nhanh do cần cập nhật lại cấu trúc phân cấp.
- Tải dồn lên nút cấp cao: dễ gây quá tải nếu nhiều yêu cầu dồn vào node trung gian.

Ví dụ:

- Hệ thống DNS: phân cấp theo tên miền (root → .com → example.com).
- Mạng tổ chức nội bộ client-server.

2. Kiến trúc ngang hàng (Peer-to-Peer - P2P Architecture)

Ưu điểm:

- Không có điểm lỗi trung tâm: mỗi node vừa là client vừa là server → hệ thống bền hơn trước lỗi cục bộ.
- Tự động mở rộng tốt: dễ thêm node mới mà không ảnh hưởng lớn đến cấu trúc chung.

- Phân phối tải hiệu quả: các node cùng chia sẻ tài nguyên và xử lý.

Nhược điểm:

- Khó quản lý và điều phối: không có điểm kiểm soát trung tâm → khó giám sát toàn hệ thống.
- Bảo mật và tin cậy kém hơn: do các node là người dùng bình thường, hệ thống dễ bị tấn công hoặc có hành vi xấu.
- Tìm kiếm không hiệu quả trong mạng không cấu trúc: cần broadcast hoặc flooding gây tốn tài nguyên.

Ví dụ:

- BitTorrent: chia sẻ file giữa các người dùng.
- Skype (phiên bản cũ): mạng gọi thoại P2P không dùng server trung tâm.

Bảng so sánh tổng quát:

Tiêu chí	Phân cấp (Hierarchical)	Ngang hàng (Peer-to-Peer)
Cấu trúc quản lý	Tập trung, có phân tầng rõ ràng	Phi tập trung, các node ngang hàng
Độ tin cậy	Thấp nếu node trung tâm lỗi	Cao hơn, do không phụ thuộc 1 node
Hiệu năng tìm kiếm	Nhanh, điều phối rõ	Tùy theo cấu trúc (chậm với unstructured)
Dễ mở rộng	Trung bình – khó mở rộng động	Rất dễ mở rộng
Quản lý & kiểm soát	Dễ giám sát, bảo mật tốt hơn	Khó kiểm soát, dễ bị lạm dụng
Ứng dụng phù hợp	Doanh nghiệp, tổ chức	Chia sẻ dữ liệu, mạng cộng đồng

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?

Dưới đây là phần trình bày rõ ràng về bốn mô hình hệ thống phân tán, kèm ví dụ ứng dụng điển hình cho từng mô hình:

1. Mô hình phân tầng (Layered Model)

Khái niệm:

- Hệ thống được chia thành nhiều tầng (layers), mỗi tầng cung cấp dịch vụ cho tầng trên và sử dụng dịch vụ của tầng dưới.
- Giao tiếp chủ yếu là top-down hoặc bottom-up.

Đặc điểm:

- Cấu trúc rõ ràng, dễ bảo trì và phát triển.
- Dễ dàng thay thế hoặc nâng cấp một tầng mà không ảnh hưởng toàn bộ hệ thống.

Ví dụ ứng dụng:

- Web Application với mô hình 3-tier:
  - Tầng trình bày (Frontend: trình duyệt)
  - Tầng nghiệp vụ (Backend server)
  - Tầng cơ sở dữ liệu (Database server)

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

Khái niệm:

- Các thành phần hệ thống được xây dựng dưới dạng đối tượng, có thể gọi phương thức từ xa (Remote Method Invocation - RMI) như gọi hàm cục bộ.
- Hỗ trợ che giấu chi tiết phân tán khỏi lập trình viên.

Đặc điểm:

- Hướng đối tượng, phù hợp với mô hình lập trình hiện đại.
- Phụ thuộc mạnh vào middleware để xử lý định vị, giao tiếp, bảo mật.

Ví dụ ứng dụng:

- Java RMI, CORBA, .NET Remoting
- Hệ thống quản lý ngân hàng hoặc quản lý sinh viên dùng gọi hàm giữa các máy chủ.

3. Mô hình kênh sự kiện (Event-based Model / Publish-Subscribe)

Khái niệm:

- Các thành phần giao tiếp thông qua các sự kiện (events).
- Có publisher (phát sự kiện) và subscriber (nghe sự kiện), không cần biết nhau trực tiếp.

Đặc điểm:

- Lỏng lẻo, dễ mở rộng và tích hợp.
- Phù hợp với các hệ thống động, có nhiều tác nhân tương tác không đồng bộ.

Ví dụ ứng dụng:

- Hệ thống cảnh báo thời gian thực
  - Ví dụ: khi cảm biến nhiệt độ vượt ngưỡng, sẽ phát sự kiện cho các ứng dụng nghe (như gửi email cảnh báo).
- Kafka, MQTT, RabbitMQ

4. Mô hình dữ liệu tập trung (Data-Centered Model)

Khái niệm:

- Các thành phần hệ thống giao tiếp thông qua một kho dữ liệu chung (shared data repository).
- Các tiến trình đọc/ghi dữ liệu vào một vị trí tập trung.

Đặc điểm:

- Dễ quản lý và kiểm soát dữ liệu.
- Có thể gây nghẽn nếu nhiều tiến trình truy cập đồng thời.

Ví dụ ứng dụng:

- Hệ thống bảng đen (Blackboard system) trong trí tuệ nhân tạo.
- Cơ sở dữ liệu trung tâm trong hệ thống thông tin bệnh viện – nơi tất cả máy trạm đều đọc/ghi thông tin bệnh nhân vào một DB.

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?

Vai trò của phần mềm trung gian (middleware) trong kiến trúc khách – chủ phân tán

1. Định nghĩa:

Middleware là lớp phần mềm trung gian nằm giữa hệ điều hành và ứng dụng trong hệ thống phân tán.

Trong kiến trúc khách – chủ (client-server), middleware đóng vai trò cầu nối giữa các thành phần khách (client) và chủ (server), giúp hai phía giao tiếp hiệu quả, ẩn đi sự phức tạp của phân tán.

2. Vai trò chính của middleware:

a. Ẩn chi tiết phân tán (distribution transparency)

- Giúp client không cần biết server ở đâu, giao tiếp ra sao.
- Ví dụ: gọi phương thức từ xa như gọi hàm cục bộ (RMI, gRPC).

b. Quản lý giao tiếp giữa các tiến trình

- Thiết lập, duy trì và đồng bộ kết nối giữa client và server.
- Xử lý truyền dữ liệu, định dạng thông điệp, mã hóa/gỡ lỗi, v.v.

c. Hỗ trợ tích hợp hệ thống dị thể

- Cho phép các thành phần dùng ngôn ngữ, hệ điều hành khác nhau tương tác với nhau.
- Ví dụ: Java client giao tiếp với server viết bằng C++ thông qua middleware chuẩn (như CORBA).

3. Ba tính năng chính mà middleware cung cấp

Tính năng	Mô tả
1. Giao tiếp minh bạch	Cung cấp API/đối tượng để client gọi phương thức từ xa (RPC, RMI…)
2. Định vị và kết nối động	Cho phép tìm và kết nối tới server tại thời điểm chạy
3. Quản lý dịch vụ hệ thống	Hỗ trợ bảo mật, cân bằng tải, xử lý lỗi, sao lưu, xác thực…

Ví dụ về middleware phổ biến:

- CORBA (Common Object Request Broker Architecture)

- Java RMI (Remote Method Invocation)
- gRPC (Google Remote Procedure Call)
- MQTT, RabbitMQ (middleware hướng sự kiện)
- Apache Kafka (event-streaming)

Kết luận:

Middleware là xương sống trong kiến trúc khách – chủ phân tán, giúp ẩn phức tạp, kết nối linh hoạt và tăng khả năng tương tác giữa các hệ thống không đồng nhất.

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?

Dưới đây là phần trình bày rõ ràng về ba loại dịch vụ trong kiến trúc hướng dịch vụ (SOA – Service-Oriented Architecture), kèm ví dụ điển hình:

1. Dịch vụ cơ bản (Basic Services)

Khái niệm:

- Là các chức năng nguyên thủy, thực hiện một nhiệm vụ cụ thể, thường là tác vụ đơn lẻ trong hệ thống.

Ví dụ:

- Kiểm tra tồn kho (CheckInventoryService)
- Lấy thông tin khách hàng (GetCustomerInfoService)
- Tính thuế (CalculateTaxService)

Đặc điểm:

- Dễ tái sử dụng
- Không phụ thuộc vào ngữ cảnh quy trình lớn hơn
- Là khối xây dựng nền tảng cho các dịch vụ cao hơn

2. Dịch vụ tích hợp (Composite / Integration Services)

Khái niệm:

- Là dịch vụ kết hợp nhiều dịch vụ cơ bản lại để thực hiện một tác vụ phức tạp hơn.
- Thường thực hiện xâu chuỗi, điều phối hoặc gom dữ liệu từ nhiều dịch vụ cơ bản.

Ví dụ:

- Xử lý đơn hàng (OrderProcessingService): gọi lần lượt các dịch vụ như kiểm tra hàng, tính giá, tạo đơn hàng.
- Tạo hồ sơ khách hàng mới (CreateCustomerProfileService): kết hợp lấy thông tin, xác thực, và lưu vào hệ thống.

Đặc điểm:

- Đóng vai trò như trung gian điều phối
- Cần quản lý thứ tự gọi, xử lý lỗi trung gian

- Có thể gọi theo workflow hoặc orchestration

3. Dịch vụ quy trình (Process Services)

Khái niệm:

- Là dịch vụ đại diện cho toàn bộ một quy trình nghiệp vụ (business process).
- Gắn liền với chiến lược hoặc logic nghiệp vụ của tổ chức.

Ví dụ:

- Đăng ký bảo hiểm y tế: gồm xác minh thông tin, tính phí, tạo hợp đồng, gửi xác nhận.

Đặc điểm:

- Phức tạp nhất, gắn với quy trình toàn vẹn
- Thường được mô hình hóa bằng ngôn ngữ như BPEL, BPMN
- Có thể kéo dài qua nhiều hệ thống, nhiều ngày

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?

Vòng đời của một dịch vụ SOA gồm 4 giai đoạn chính:

1. Thiết kế và phát triển (Design & Development)

Nội dung:

- Xác định chức năng dịch vụ (tên, đầu vào/đầu ra, giao thức…).
- Thiết kế giao diện dịch vụ (WSDL nếu dùng SOAP, hoặc OpenAPI nếu dùng REST).
- Cài đặt logic xử lý bên trong dịch vụ.

Thách thức chính:

- Tái sử dụng vs thiết kế mới: Dễ bị trùng lặp nếu không kiểm soát tốt kho dịch vụ.
- Thiết kế giao diện không chuẩn sẽ khó tích hợp sau này.
- Đồng bộ giữa các nhóm phát triển dịch vụ và nhóm tiêu dùng dịch vụ.

2. Kiểm thử và triển khai (Testing & Deployment)

Nội dung:

- Kiểm thử đơn vị (unit test), kiểm thử tích hợp, hiệu năng, bảo mật.
- Đóng gói dịch vụ và triển khai lên môi trường staging hoặc production.
- Cấu hình kết nối, bảo mật, giám sát ban đầu.

Thách thức chính:

- Kiểm thử khó khăn khi dịch vụ phụ thuộc vào dịch vụ khác (cần mock/fake service).
- Thiếu tự động hóa trong triển khai (CI/CD).
- Kiểm soát phiên bản (versioning) – các dịch vụ tiêu dùng cũ có thể bị ảnh hưởng.

3. Công bố và khám phá (Publishing & Discovery)

Nội dung:

- Đăng ký dịch vụ vào Service Registry (như UDDI, Consul, Eureka…).
- Cho phép các hệ thống khác tìm kiếm và sử dụng dịch vụ.

Thách thức chính:

- Registry lỗi thời hoặc không đồng bộ → dịch vụ tồn tại nhưng không ai biết.
- Quản lý quyền truy cập (ai được gọi dịch vụ nào).
- Thiếu metadata mô tả đầy đủ về dịch vụ (version, SLA, liên hệ...).

4. Vận hành và giám sát (Operation & Monitoring)

Nội dung:

- Đảm bảo dịch vụ hoạt động ổn định trong môi trường thật.
- Giám sát hiệu năng, lỗi, bảo mật, xử lý sự cố.
- Quản lý thay đổi và cập nhật dịch vụ theo thời gian.

Thách thức chính:

- Khó phát hiện nguyên nhân lỗi trong hệ thống nhiều dịch vụ (microservices).
- Phải đảm bảo tính sẵn sàng cao và thời gian phản hồi thấp.
- Giám sát phân tán đòi hỏi công cụ mạnh (Prometheus, ELK stack, Zipkin...).