**CÁC HỆ THỐNG PHÂN TÁN**

**Nhóm 13:**

* **Cao Hải Long – B24CHHT081**
* **Nguyễn Anh Đức – B24CHHT063**
* **Nguyễn Tiến Anh – B24CHHT055**

**ĐỀ BÀI SESSION 1:**

**Câu 1:**

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

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

**Câu 2:**

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.

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?

**Câu 3:**

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

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

**Câu 4:**

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

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.

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

**Câu 5:**

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

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.

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.

**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.

**TRẢ LỜI:**

**Câu 1:**

**1. Hai đặc điểm quan trọng nhất của hệ thống phân tán và giải thích**

Hệ thống phân tán là hệ thống mà các thành phần (nút, máy tính) hoạt động độc lập, phối hợp qua mạng để đạt được mục tiêu chung. Hai đặc điểm quan trọng nhất của hệ thống phân tán là:

**a. Không có đồng hồ chung (No Global Clock):**

* **Giải thích**: Trong hệ thống phân tán, mỗi nút có đồng hồ riêng và không có một đồng hồ toàn cục để đồng bộ hóa thời gian giữa các nút. Điều này dẫn đến khó khăn trong việc xác định thứ tự các sự kiện xảy ra trên các nút khác nhau. Ví dụ, khi hai nút gửi tin nhắn, không thể xác định chính xác tin nhắn nào được gửi trước nếu không có cơ chế đồng bộ hóa thời gian (như Lamport Timestamp hoặc Vector Clock).
* **Tầm quan trọng**: Việc thiếu đồng hồ chung ảnh hưởng đến các tác vụ như đồng bộ hóa, lập lịch, hoặc đảm bảo tính nhất quán dữ liệu. Các hệ thống phân tán phải sử dụng các thuật toán đặc biệt (như NTP hoặc các giao thức đồng bộ) để xử lý vấn đề này.

**b. Các thành phần hoạt động độc lập và có khả năng thất bại độc lập (Independent Failure):**

* **Giải thích**: Mỗi nút trong hệ thống phân tán hoạt động độc lập, nghĩa là một nút có thể gặp lỗi (crash, mất kết nối, chậm trễ) mà không làm toàn bộ hệ thống ngừng hoạt động. Tuy nhiên, điều này đòi hỏi hệ thống phải có cơ chế phát hiện và xử lý lỗi (fault tolerance) như sao lưu dữ liệu, chuyển đổi dự phòng, hoặc cơ chế đồng thuận (consensus).
* **Tầm quan trọng**: Đặc điểm này giúp hệ thống phân tán có tính khả dụng cao, nhưng đồng thời làm tăng độ phức tạp trong việc thiết kế các cơ chế để xử lý lỗi cục bộ mà không ảnh hưởng đến toàn hệ thống.

**2. Ba lý do cơ bản khiến các ứng dụng phân tán phức tạp hơn so với ứng dụng đơn lẻ**

Ứng dụng phân tán phức tạp hơn ứng dụng đơn lẻ do phải xử lý các vấn đề liên quan đến sự phân bố và phối hợp giữa nhiều nút. Dưới đây là ba lý do chính:

**a. Giao tiếp qua mạng và độ trễ (Network Communication and Latency):**

* Trong ứng dụng đơn lẻ, các thành phần giao tiếp trực tiếp trong cùng một máy, tốc độ nhanh và đáng tin cậy. Trong ứng dụng phân tán, các nút giao tiếp qua mạng, dẫn đến độ trễ (latency), mất gói tin (packet loss), hoặc ngắt kết nối. Điều này đòi hỏi các cơ chế như giao thức truyền tin đáng tin cậy, xử lý lỗi mạng, và tối ưu hóa hiệu suất giao tiếp.
* **Ví dụ**: Một ứng dụng web phân tán cần xử lý các yêu cầu từ client đến nhiều server, trong khi ứng dụng đơn lẻ chỉ cần xử lý cục bộ.

**b. Tính nhất quán dữ liệu (Data Consistency):**

* Ứng dụng phân tán thường lưu trữ dữ liệu trên nhiều nút, dẫn đến thách thức trong việc đảm bảo tính nhất quán (consistency) giữa các bản sao dữ liệu. Các mô hình như eventual consistency, strong consistency, hoặc causal consistency đòi hỏi các thuật toán phức tạp như Paxos, Raft, hoặc CRDT (Conflict-free Replicated Data Types).
* Trong ứng dụng đơn lẻ, dữ liệu được quản lý trong một hệ thống duy nhất, không cần lo lắng về vấn đề đồng bộ hóa giữa các bản sao.

**c. Xử lý lỗi và tính khả dụng (Fault Tolerance and Availability):**

* Các ứngRainbow thể hiện sự cố gắng khắc phục lỗi của hệ thống phân tán phức tạp hơn do các nút có thể thất bại độc lập. Điều này đòi hỏi các cơ chế phát hiện lỗi, sao lưu, và khôi phục hệ thống để đảm bảo tính khả dụng. Trong khi đó, ứng dụng đơn lẻ chạy trên một máy duy nhất, nên lỗi phần cứng hoặc phần mềm thường làm toàn bộ hệ thống ngừng hoạt động, đơn giản hóa việc xử lý lỗi.
* **Ví dụ**: Trong hệ thống phân tán như cơ sở dữ liệu NoSQL (Cassandra), cần đảm bảo dữ liệu được sao chép và đồng bộ giữa các nút, ngay cả khi một số nút gặp sự cố.

**Câu 2:**

**1. Phân tích các yếu tố phần cứng và mạng ảnh hưởng đến hiệu năng hệ thống phân tán**

Hiệu năng của hệ thống phân tán phụ thuộc vào nhiều yếu tố phần cứng và mạng, bao gồm CPU, bộ nhớ, kênh truyền, băng thông, và topology mạng. Dưới đây là phân tích chi tiết:

**\*) Yếu tố phần cứng**

**a. CPU (Bộ xử lý):**

* **Ảnh hưởng**: CPU quyết định tốc độ xử lý các tác vụ tính toán tại mỗi nút trong hệ thống phân tán. Trong hệ thống phân tán, các nút thường thực hiện các tác vụ độc lập hoặc phối hợp, vì vậy sức mạnh CPU của từng nút ảnh hưởng trực tiếp đến thời gian xử lý cục bộ và hiệu suất tổng thể. Nếu một nút có CPU yếu, nó có thể trở thành nút thắt cổ chai (bottleneck), làm chậm toàn bộ hệ thống.
* **Ví dụ**: Trong một cụm tính toán phân tán (như Hadoop), CPU yếu trên một nút có thể làm chậm quá trình xử lý các tác vụ MapReduce.
* **Giải pháp**: Sử dụng CPU đa lõi, tối ưu hóa thuật toán, hoặc cân bằng tải để phân phối công việc đều giữa các nút.

**b. Bộ nhớ (Memory):**

* **Ảnh hưởng**: Dung lượng và tốc độ bộ nhớ (RAM) ảnh hưởng đến khả năng lưu trữ dữ liệu tạm thời và xử lý các tác vụ lớn. Trong hệ thống phân tán, các nút thường cần lưu trữ bản sao dữ liệu hoặc trạng thái cục bộ, nên bộ nhớ không đủ có thể dẫn đến việc truy xuất đĩa (I/O bottleneck), làm giảm hiệu năng. Ngoài ra, độ trễ truy cập bộ nhớ cũng ảnh hưởng đến tốc độ xử lý.
* **Ví dụ**: Trong cơ sở dữ liệu phân tán như Cassandra, bộ nhớ lớn giúp lưu trữ dữ liệu trong bộ nhớ cache, giảm thời gian truy xuất từ đĩa.
* **Giải pháp**: Tăng dung lượng RAM, sử dụng bộ nhớ cache hiệu quả, hoặc áp dụng các kỹ thuật nén dữ liệu.

**c. Kênh truyền (I/O Channels):**

* **Ảnh hưởng**: Kênh truyền (như ổ cứng, SSD, hoặc các thiết bị lưu trữ) quyết định tốc độ đọc/ghi dữ liệu tại mỗi nút. Trong hệ thống phân tán, dữ liệu thường được truyền qua mạng hoặc lưu trữ cục bộ, nên tốc độ kênh truyền ảnh hưởng đến hiệu suất tổng thể, đặc biệt trong các ứng dụng yêu cầu I/O cao như cơ sở dữ liệu hoặc phân tích big data.
* **Ví dụ**: Một hệ thống tệp phân tán (như HDFS) phụ thuộc vào tốc độ đọc/ghi của ổ đĩa để lưu trữ và truy xuất khối dữ liệu lớn.
* **Giải pháp**: Sử dụng SSD thay vì HDD, tối ưu hóa truy xuất I/O, hoặc sử dụng bộ nhớ đệm để giảm tải kênh truyền.

**\*) Yếu tố mạng**

**a. Băng thông (Bandwidth):**

* **Ảnh hưởng**: Băng thông quyết định lượng dữ liệu có thể truyền qua mạng trong một đơn vị thời gian. Trong hệ thống phân tán, các nút liên tục trao đổi dữ liệu (như tin nhắn, bản sao dữ liệu, hoặc trạng thái), nên băng thông thấp có thể gây tắc nghẽn mạng, làm tăng độ trễ và giảm hiệu năng. Băng thông cao giúp cải thiện tốc độ truyền dữ liệu và khả năng mở rộng của hệ thống.
* **Ví dụ**: Trong một hệ thống streaming như Apache Kafka, băng thông thấp có thể làm chậm việc truyền dữ liệu giữa các broker.
* **Giải pháp**: Sử dụng mạng tốc độ cao (như 10GbE), nén dữ liệu trước khi truyền, hoặc tối ưu hóa giao thức truyền thông.

**b. Topology mạng (Network Topology):**

* **Ảnh hưởng**: Topology mạng (như star, mesh, ring, hoặc tree) xác định cách các nút được kết nối và ảnh hưởng đến hiệu suất truyền thông. Topology hiệu quả giảm độ trễ và tăng khả năng chịu lỗi, trong khi topology kém tối ưu có thể gây tắc nghẽn hoặc lỗi lan truyền. Ví dụ, trong topology star, nếu nút trung tâm thất bại, toàn bộ hệ thống có thể bị ảnh hưởng.
* **Ví dụ**: Trong một cụm tính toán, topology fat-tree thường được sử dụng để giảm tắc nghẽn và tăng băng thông giữa các nút.
* **Giải pháp**: Lựa chọn topology phù hợp với ứng dụng (như mesh cho hệ thống P2P), đảm bảo dự phòng kết nối, và sử dụng các thiết bị mạng hiệu suất cao.

**2. Tại sao hệ điều hành phân tán (Distributed OS) và hệ điều hành mạng (Network OS) có yêu cầu khác nhau về quản lý tài nguyên?**

Hệ điều hành phân tán (Distributed OS) và hệ điều hành mạng (Network OS) có mục đích và cách hoạt động khác nhau, dẫn đến các yêu cầu quản lý tài nguyên khác nhau. Dưới đây là các lý do chính:

**a. Mức độ tích hợp và tính minh bạch (Transparency):**

* **Distributed OS**: Được thiết kế để quản lý một hệ thống phân tán như một hệ thống duy nhất, cung cấp tính minh bạch cao (transparent) về tài nguyên. Người dùng không cần biết tài nguyên (CPU, bộ nhớ, lưu trữ) nằm ở đâu, vì hệ điều hành tự động phân phối và quản lý tài nguyên trên nhiều nút. Điều này đòi hỏi cơ chế phức tạp để đồng bộ hóa, lập lịch tài nguyên toàn cục, và đảm bảo tính nhất quán.
  + **Yêu cầu**: Quản lý tài nguyên toàn cục (global resource management), đồng bộ hóa chặt chẽ, và các thuật toán như đồng thuận (consensus) hoặc lập lịch phân tán.
  + **Ví dụ**: Hệ điều hành Amoeba hoặc Sprite, nơi tài nguyên được quản lý như một hệ thống thống nhất.
* **Network OS**: Cung cấp giao tiếp giữa các máy tính độc lập, nhưng mỗi máy vẫn giữ hệ điều hành riêng và quản lý tài nguyên cục bộ. Người dùng phải biết rõ tài nguyên nằm trên nút nào (ít minh bạch hơn). Network OS chỉ hỗ trợ giao tiếp và chia sẻ tài nguyên cơ bản qua mạng.
  + **Yêu cầu**: Quản lý tài nguyên cục bộ tại mỗi nút, với các giao thức mạng để chia sẻ tài nguyên (như NFS cho tệp hoặc RPC cho gọi hàm từ xa).
  + **Ví dụ**: UNIX với NFS hoặc Windows với SMB.

**b. Mức độ kiểm soát tài nguyên:**

* **Distributed OS**: Yêu cầu kiểm soát tài nguyên chặt chẽ trên toàn hệ thống để đảm bảo hiệu suất và tính nhất quán. Ví dụ, một Distributed OS có thể phân bổ CPU hoặc bộ nhớ từ nhiều nút để thực hiện một tác vụ duy nhất, đòi hỏi cơ chế lập lịch toàn cục và xử lý lỗi phân tán.
  + **Yêu cầu**: Các thuật toán phức tạp như load balancing, fault tolerance, và đồng bộ hóa dữ liệu trên nhiều nút.
* **Network OS**: Chỉ kiểm soát tài nguyên cục bộ tại mỗi nút, với sự phối hợp tối thiểu giữa các nút. Ví dụ, một nút có thể chia sẻ tệp qua mạng, nhưng không cần quản lý tài nguyên của nút khác.
  + **Yêu cầu**: Quản lý tài nguyên đơn giản hơn, tập trung vào giao tiếp mạng và chia sẻ tài nguyên cơ bản.

**c. Mục tiêu và độ phức tạp:**

* **Distributed OS**: Nhằm cung cấp một môi trường thống nhất, che giấu sự phức tạp của hệ thống phân tán khỏi người dùng. Điều này đòi hỏi các cơ chế quản lý tài nguyên phức tạp hơn, như đồng bộ hóa thời gian, quản lý bộ nhớ phân tán, hoặc xử lý lỗi toàn cục.
  + **Ví dụ**: Quản lý bộ nhớ phân tán yêu cầu cơ chế như DSM (Distributed Shared Memory) để tạo ảo giác về một không gian bộ nhớ chung.
* **Network OS**: Tập trung vào cung cấp khả năng giao tiếp và chia sẻ tài nguyên giữa các hệ thống độc lập, với mục tiêu đơn giản hơn là hỗ trợ kết nối mạng. Quản lý tài nguyên chủ yếu là cục bộ, với các giao thức mạng để truy cập tài nguyên từ xa.
  + **Ví dụ**: Một Network OS như Linux với NFS chỉ cần quản lý quyền truy cập tệp từ xa, không cần đồng bộ hóa trạng thái toàn cục.

**Câu 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**

Hệ thống phân tán bao gồm nhiều loại khác nhau, được phân loại dựa trên mục đích sử dụng và cách tổ chức. Dưới đây là ba loại chính: **điện toán phân tán**, **thông tin phân tán**, và **lan tỏa phân tán**, cùng với so sánh chi tiết:

**a. Điện toán phân tán (Distributed Computing Systems)**

* **Định nghĩa**: Hệ thống mà các nút hợp tác để thực hiện các tác vụ tính toán lớn, thường được chia nhỏ thành các tác vụ con và thực hiện song song trên nhiều máy tính. Mục tiêu là tối ưu hóa hiệu suất tính toán.
* **Đặc điểm**:
  + Tập trung vào xử lý tính toán, chẳng hạn như phân tích dữ liệu lớn, mô phỏng khoa học, hoặc render đồ họa.
  + Các nút thường đồng bộ hóa chặt chẽ để đảm bảo kết quả tính toán chính xác.
  + Ví dụ: Cụm tính toán (clusters) như Hadoop, MPI (Message Passing Interface), hoặc các hệ thống siêu máy tính.
* **Ứng dụng**: Phân tích dữ liệu lớn (Big Data), mô phỏng vật lý, học máy phân tán.

**b. Thông tin phân tán (Distributed Information Systems)**

* **Định nghĩa**: Hệ thống tập trung vào việc lưu trữ, truy xuất, và chia sẻ thông tin/dữ liệu giữa các nút. Dữ liệu thường được phân phối trên nhiều máy chủ để đảm bảo tính khả dụng và hiệu suất.
* **Đặc điểm**:
  + Tập trung vào quản lý dữ liệu, đảm bảo tính nhất quán và khả năng truy cập.
  + Sử dụng các giao thức như giao dịch phân tán (distributed transactions) hoặc cơ chế đồng bộ dữ liệu.
  + Ví dụ: Cơ sở dữ liệu phân tán (như Cassandra, MongoDB), hệ thống ERP (Enterprise Resource Planning), hoặc hệ thống quản lý giao dịch ngân hàng.
* **Ứng dụng**: Hệ thống ngân hàng, thương mại điện tử, quản lý dữ liệu doanh nghiệp.

**c. Lan tỏa phân tán (Pervasive Distributed Systems)**

* **Định nghĩa**: Hệ thống mà các thiết bị tính toán (thường là thiết bị nhỏ, nhúng) được tích hợp vào môi trường xung quanh, hoạt động liên tục và tương tác với người dùng một cách tự nhiên. Mục tiêu là tạo ra môi trường thông minh, liền mạch.
* **Đặc điểm**:
  + Các thiết bị thường có tài nguyên hạn chế (CPU, bộ nhớ, năng lượng), như cảm biến IoT, thiết bị di động, hoặc thiết bị gia dụng thông minh.
  + Hoạt động trong môi trường động, không ổn định, với các thiết bị có thể tham gia hoặc rời đi bất kỳ lúc nào.
  + Ví dụ: Nhà thông minh, hệ thống IoT (Internet of Things), mạng cảm biến không dây.
* **Ứng dụng**: Thành phố thông minh, y tế thông minh, giám sát môi trường.

**So sánh ba loại hệ thống phân tán**

| **Tiêu chí** | **Điện toán phân tán** | **Thông tin phân tán** | **Lan tỏa phân tán** |
| --- | --- | --- | --- |
| **Mục tiêu chính** | Tối ưu hóa hiệu suất tính toán | Quản lý và chia sẻ dữ liệu | Tích hợp vào môi trường, tương tác liền mạch |
| **Tài nguyên** | Máy tính mạnh, cụm tính toán | Máy chủ, cơ sở dữ liệu | Thiết bị nhúng, cảm biến, tài nguyên hạn chế |
| **Đồng bộ hóa** | Cao, yêu cầu đồng bộ chặt chẽ | Trung bình, tập trung vào nhất quán dữ liệu | Thấp, hoạt động trong môi trường động |
| **Độ phức tạp** | Cao do phân phối tác vụ tính toán | Trung bình, tập trung vào giao dịch | Cao do môi trường không ổn định |
| **Ứng dụng tiêu biểu** | Big Data, học máy, siêu tính toán | Ngân hàng, thương mại điện tử | IoT, nhà thông minh, y tế thông minh |
| **Ví dụ** | Hadoop, MPI | Cassandra, Oracle Distributed DB | Zigbee, mạng cảm biến không dây |

**Nhận xét**:

* Điện toán phân tán tập trung vào hiệu suất tính toán, thông tin phân tán ưu tiên quản lý dữ liệu, còn lan tỏa phân tán hướng đến tích hợp vào môi trường thực tế.
* Mỗi loại có yêu cầu khác nhau về tài nguyên, đồng bộ hóa, và độ phức tạp, phù hợp với các kịch bản ứng dụng riêng biệt.

**2. Phân tích các lớp chính trong kiến trúc điện toán lưới và vai trò của từng lớp**

**Điện toán lưới (Grid Computing)** là một dạng điện toán phân tán, trong đó các tài nguyên tính toán và lưu trữ từ nhiều tổ chức được chia sẻ để thực hiện các tác vụ lớn. Kiến trúc điện toán lưới thường được phân thành ba lớp chính: **Application**, **Middleware**, và **Resource**. Dưới đây là phân tích vai trò của từng lớp:

**a. Lớp ứng dụng (Application Layer)**

* **Vai trò**:
  + Là giao diện giữa người dùng và hệ thống điện toán lưới, cung cấp các ứng dụng hoặc công cụ để người dùng khai thác tài nguyên lưới.
  + Xử lý các yêu cầu của người dùng, như chạy mô phỏng khoa học, phân tích dữ liệu, hoặc render đồ họa.
  + Chuyển đổi yêu cầu người dùng thành các tác vụ có thể được phân phối và thực thi trên lưới.
* **Đặc điểm**:
  + Bao gồm các ứng dụng người dùng, cổng lưới (grid portals), hoặc các công cụ phát triển ứng dụng.
  + Tùy thuộc vào loại ứng dụng, lớp này có thể yêu cầu giao diện đồ họa hoặc dòng lệnh.
* **Ví dụ**:
  + Ứng dụng phân tích dữ liệu sinh học (như BLAST trên lưới).
  + Cổng web để nộp công việc (job submission) trên hệ thống lưới như Globus Toolkit.
* **Tầm quan trọng**: Là cầu nối trực tiếp với người dùng, đảm bảo tính dễ sử dụng và hiệu quả khai thác hệ thống.

**b. Lớp trung gian (Middleware Layer)**

* **Vai trò**:
  + Hoạt động như "bộ não" của hệ thống điện toán lưới, quản lý và điều phối các tài nguyên phân tán.
  + Cung cấp các dịch vụ như lập lịch tác vụ (scheduling), quản lý tài nguyên, bảo mật, và truyền dữ liệu.
  + Che giấu sự phức tạp của lớp tài nguyên, tạo ra một giao diện thống nhất để lớp ứng dụng truy cập tài nguyên.
* **Đặc điểm**:
  + Bao gồm các phần mềm trung gian như Globus Toolkit, Condor, hoặc BOINC.
  + Hỗ trợ các chức năng như xác thực (authentication), phân quyền (authorization), phát hiện tài nguyên (resource discovery), và xử lý lỗi.
  + Đảm bảo tính minh bạch (transparency) trong việc truy cập tài nguyên phân tán.
* **Ví dụ**:
  + Globus Toolkit cung cấp dịch vụ quản lý tài nguyên và truyền tệp trên lưới.
  + Condor lập lịch các công việc tính toán trên các tài nguyên có sẵn.
* **Tầm quan trọng**: Là lớp cốt lõi, đảm bảo sự phối hợp giữa các tài nguyên phân tán, tối ưu hóa hiệu suất và tính khả dụng.

**c. Lớp tài nguyên (Resource Layer)**

* **Vai trò**:
  + Bao gồm các tài nguyên vật lý (CPU, bộ nhớ, lưu trữ, mạng) được chia sẻ trong hệ thống điện toán lưới.
  + Cung cấp sức mạnh tính toán và lưu trữ thực tế để thực thi các tác vụ từ lớp ứng dụng.
  + Được quản lý bởi các hệ thống cục bộ (như hệ điều hành hoặc phần mềm quản lý cụm) và phối hợp qua lớp trung gian.
* **Đặc điểm**:
  + Tài nguyên thường thuộc nhiều tổ chức khác nhau, với cấu hình và chính sách sử dụng đa dạng.
  + Có thể bao gồm siêu máy tính, cụm máy tính, máy chủ, hoặc thiết bị lưu trữ.
* **Ví dụ**:
  + Một cụm tính toán tại đại học A, một siêu máy tính tại trung tâm nghiên cứu B, hoặc - Một ổ lưu trữ tại trung tâm dữ liệu C.
  + Các tài nguyên này được kết nối qua mạng tốc độ cao.
* **Tầm quan trọng**: Là nền tảng vật lý của điện toán lưới, cung cấp tài nguyên thực tế để thực hiện các tác vụ.

**Câu 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 khả năng hệ thống phân tán tiếp tục cung cấp dịch vụ đúng chức năng ngay cả khi một số thành phần gặp sự cố (như hỏng nút, mất kết nối mạng, hoặc lỗi phần mềm). Đây được xem là mục tiêu quan trọng nhất vì các lý do sau:

* **Mục đích cốt lõi của hệ thống phân tán**: Hệ thống phân tán được thiết kế để tăng độ tin cậy và khả năng chịu lỗi so với hệ thống đơn lẻ. Bằng cách phân phối tài nguyên trên nhiều nút, hệ thống có thể duy trì hoạt động khi một hoặc vài nút thất bại. Nếu tính sẵn sàng không được đảm bảo, mục tiêu chính của hệ thống phân tán sẽ bị vô hiệu hóa.
* **Yêu cầu từ ứng dụng thực tế**: Nhiều hệ thống phân tán phục vụ các ứng dụng quan trọng như thương mại điện tử (Amazon), dịch vụ đám mây (Google Cloud), hoặc ngân hàng, nơi thời gian ngừng hoạt động (downtime) gây ra tổn thất tài chính lớn hoặc mất niềm tin khách hàng. Ví dụ, một giờ downtime của Amazon có thể gây thiệt hại hàng triệu đô la.
* **Thách thức môi trường phân tán**: Hệ thống phân tán phải đối mặt với lỗi cục bộ (partial failures) và phân vùng mạng (network partitions). Tính sẵn sàng đảm bảo hệ thống vẫn hoạt động nhờ các cơ chế như sao chép dữ liệu (replication), chuyển đổi dự phòng (failover), hoặc phân tải (load balancing).
* **CAP Theorem**: Theo định lý CAP, hệ thống phân tán chỉ có thể ưu tiên hai trong ba đặc tính: tính nhất quán (consistency), tính sẵn sàng (availability), và khả năng chịu phân vùng (partition tolerance). Trong nhiều ứng dụng thời gian thực (như mạng xã hội, tìm kiếm), tính sẵn sàng được ưu tiên hơn tính nhất quán để đảm bảo người dùng luôn nhận được phản hồi, dù dữ liệu có thể tạm thời không đồng bộ (eventual consistency).
* **Trải nghiệm người dùng và kinh tế**: Người dùng mong đợi dịch vụ luôn sẵn sàng 24/7. Downtime làm giảm trải nghiệm và có thể đẩy khách hàng sang đối thủ cạnh tranh. Tính sẵn sàng vì thế trở thành yếu tố sống còn để duy trì hoạt động kinh doanh.

**Ví dụ**: Trong cơ sở dữ liệu phân tán như Cassandra, dữ liệu được sao chép trên nhiều nút. Nếu một nút thất bại, các nút khác vẫn phục vụ yêu cầu, đảm bảo tính sẵn sàng.

**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)** trong hệ thống phân tán là khả năng che giấu sự phức tạp của hệ thống, khiến người dùng hoặc nhà phát triển cảm nhận như đang làm việc với một hệ thống đơn lẻ. Ba hình thức chính là trong suốt về truy nhập, vị trí, và lỗi. Dưới đây là giải thích, so sánh, và ví dụ minh họa:

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

* **Định nghĩa**: Người dùng có thể truy cập tài nguyên phân tán (như tệp, dữ liệu, hoặc dịch vụ) theo cách giống hệt như truy cập tài nguyên cục bộ, mà không cần biết chi tiết về giao thức hoặc định dạng dữ liệu.
* **Đặc điểm**: Che giấu sự khác biệt về cách truy cập (như giao thức mạng, định dạng dữ liệu) giữa các nút. Người dùng sử dụng cùng cú pháp hoặc giao diện.
* **Ví dụ**: Trong hệ thống tệp phân tán NFS (Network File System), người dùng có thể đọc/ghi tệp trên máy chủ từ xa giống như tệp cục bộ, mà không cần biết tệp được lưu trữ ở đâu hay giao thức truyền tải.
* **Ứng dụng**: Hệ thống tệp phân tán, cơ sở dữ liệu phân tán.

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

* **Định nghĩa**: Người dùng có thể truy cập tài nguyên mà không cần biết vị trí vật lý của tài nguyên đó (nút nào, máy chủ nào, hoặc trung tâm dữ liệu nào).
* **Đặc điểm**: Che giấu thông tin về vị trí địa lý hoặc logic của tài nguyên. Hệ thống tự động định tuyến yêu cầu đến nút chứa tài nguyên.
* **Ví dụ**: Khi truy cập một trang web như Google, người dùng nhập URL ([www.google.com](http://www.google.com" \t "_blank)) mà không cần biết máy chủ cụ thể nào (ở Mỹ, châu Âu, hay châu Á) đang xử lý yêu cầu. Hệ thống DNS và CDN đảm bảo định tuyến đúng.
* **Ứng dụng**: Web, dịch vụ đám mây, cơ sở dữ liệu phân tán.

**c. Trong suốt về lỗi (Failure Transparency)**

* **Định nghĩa**: Hệ thống che giấu các lỗi cục bộ (như hỏng nút, mất kết nối) để người dùng không nhận ra sự cố và vẫn nhận được dịch vụ bình thường.
* **Đặc điểm**: Yêu cầu cơ chế xử lý lỗi như sao chép dữ liệu, chuyển đổi dự phòng, hoặc thử lại (retry). Người dùng không bị gián đoạn dù có lỗi xảy ra.
* **Ví dụ**: Trong hệ thống thương mại điện tử như Amazon, nếu một máy chủ xử lý giỏ hàng gặp sự cố, hệ thống tự động chuyển yêu cầu sang máy chủ khác, và người dùng vẫn hoàn tất mua sắm mà không biết có lỗi.
* **Ứng dụng**: Dịch vụ đám mây, hệ thống giao dịch ngân hàng.

**So sánh ba hình thức trong suốt**

| **Tiêu chí** | **Trong suốt về truy nhập** | **Trong suốt về vị trí** | **Trong suốt về lỗi** |
| --- | --- | --- | --- |
| **Mục tiêu** | Che giấu cách truy cập tài nguyên | Che giấu vị trí vật lý của tài nguyên | Che giấu lỗi cục bộ |
| **Đối tượng che giấu** | Giao thức, định dạng dữ liệu | Vị trí địa lý/logic của tài nguyên | Sự cố phần cứng/phần mềm/mạng |
| **Cơ chế hỗ trợ** | Giao diện thống nhất, chuẩn hóa API | DNS, định tuyến, bản đồ tài nguyên | Sao chép, chuyển đổi dự phòng, thử lại |
| **Độ phức tạp** | Trung bình (phụ thuộc giao diện) | Thấp (phụ thuộc định tuyến) | Cao (phụ thuộc xử lý lỗi) |
| **Ví dụ** | NFS đọc/ghi tệp từ xa | Truy cập Google qua URL | Amazon xử lý giỏ hàng khi máy chủ lỗi |

**Nhận xét**:

* Cả ba loại trong suốt đều nhằm đơn giản hóa trải nghiệm người dùng, nhưng chúng tập trung vào các khía cạnh khác nhau: truy nhập (cách thức), vị trí (nơi lưu trữ), và lỗi (khả năng chịu lỗi).
* Trong suốt về lỗi thường phức tạp nhất vì đòi hỏi cơ chế xử lý lỗi phân tán, trong khi trong suốt về vị trí thường dễ triển khai hơn nhờ các công nghệ như DNS.

**3. Trình bày mối quan hệ giữa “tính mở” (openness) và khả năng tương tác (interoperability) trong hệ thống phân tán**

**Tính mở (openness)** và **khả năng tương tác (interoperability)** là hai đặc tính quan trọng trong hệ thống phân tán, có mối quan hệ chặt chẽ và bổ trợ lẫn nhau. Dưới đây là phân tích mối quan hệ giữa chúng:

**a. Định nghĩa**

* **Tính mở (Openness)**: Là khả năng của hệ thống phân tán được thiết kế với các thành phần dễ dàng mở rộng, sửa đổi, hoặc tích hợp với các hệ thống khác. Tính mở thường dựa trên việc sử dụng các chuẩn mở (open standards), giao diện công khai (public interfaces), và mã nguồn mở (open source) để khuyến khích sự tham gia của nhiều nhà phát triển hoặc tổ chức.
  + **Ví dụ**: Hệ thống sử dụng giao thức HTTP hoặc REST API để cho phép các ứng dụng khác tích hợp dễ dàng.
* **Khả năng tương tác (Interoperability)**: Là khả năng các thành phần hoặc hệ thống khác nhau (có thể từ các nhà cung cấp khác nhau) hoạt động cùng nhau một cách hiệu quả thông qua các giao diện và giao thức chung. Tương tác đảm bảo các hệ thống trao đổi dữ liệu và thực hiện chức năng một cách liền mạch.
  + **Ví dụ**: Một ứng dụng chạy trên Linux có thể giao tiếp với cơ sở dữ liệu Oracle chạy trên Windows thông qua giao thức ODBC.

**b. Mối quan hệ giữa tính mở và khả năng tương tác**

* **Tính mở là nền tảng cho khả năng tương tác**:
  + Tính mở cung cấp các chuẩn mở, giao diện công khai, và tài liệu rõ ràng, tạo điều kiện cho các hệ thống khác nhau tích hợp và giao tiếp với nhau. Nếu một hệ thống không mở (ví dụ, sử dụng giao thức độc quyền), khả năng tương tác sẽ bị hạn chế.
  + Ví dụ: Giao thức HTTP là một chuẩn mở, cho phép các trình duyệt (Chrome, Firefox) và máy chủ (Apache, Nginx) từ các nhà cung cấp khác nhau tương tác dễ dàng.
* **Khả năng tương tác là mục tiêu của tính mở**:
  + Tính mở không chỉ dừng ở việc công khai giao diện mà còn hướng đến việc đảm bảo các hệ thống khác nhau có thể làm việc cùng nhau (tức là đạt được khả năng tương tác). Một hệ thống mở nhưng không tương tác được với hệ thống khác sẽ không phát huy được giá trị.
  + Ví dụ: Các dịch vụ đám mây như AWS và Azure hỗ trợ chuẩn Kubernetes, cho phép ứng dụng triển khai trên cả hai nền tảng mà không cần sửa đổi lớn.
* **Tương hỗ lẫn nhau**:
  + Tính mở thúc đẩy khả năng tương tác bằng cách cung cấp các tiêu chuẩn chung (như XML, JSON, SOAP). Ngược lại, khả năng tương tác khuyến khích các nhà phát triển áp dụng tính mở để đảm bảo hệ thống của họ tương thích với các hệ thống khác, từ đó tăng khả năng cạnh tranh và áp dụng rộng rãi.
  + Ví dụ: Hệ thống nhắn tin phân tán như Matrix sử dụng giao thức mở, cho phép các ứng dụng như Element và Riot tương tác với nhau, đồng thời tích hợp với các nền tảng khác như Slack.

**c. Tác động trong hệ thống phân tán**

* **Lợi ích của sự kết hợp**:
  + **Tính mở** giúp hệ thống dễ dàng mở rộng (scalability) và tích hợp với các công nghệ mới, trong khi **khả năng tương tác** đảm bảo các thành phần phân tán (như nút, dịch vụ, hoặc ứng dụng) hoạt động hài hòa.
  + Ví dụ: Trong điện toán lưới (Grid Computing), các tài nguyên từ nhiều tổ chức sử dụng chuẩn Globus Toolkit (mở) để tương tác, đảm bảo các hệ thống khác nhau phối hợp hiệu quả.
* **Thách thức**:
  + Đảm bảo tính mở đòi hỏi chi phí thiết kế và tài liệu hóa giao diện, trong khi khả năng tương tác yêu cầu các nhà cung cấp tuân thủ nghiêm ngặt các chuẩn chung, điều này có thể khó thực hiện nếu có sự khác biệt về mục tiêu hoặc lợi ích kinh doanh.
  + Ví dụ: Một số hệ thống độc quyền (proprietary systems) như phần mềm ERP của SAP, có thể hạn chế tính mở, làm giảm khả năng tương tác với các hệ thống không phải của SAP.

**Câu 5:**

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

**a. Kiến trúc phân cấp (Client-Server Architecture)**

* **Định nghĩa**: Trong kiến trúc phân cấp, hệ thống được tổ chức thành các máy khách (clients) gửi yêu cầu và các máy chủ (servers) cung cấp dịch vụ hoặc tài nguyên. Các máy chủ thường có vai trò trung tâm, xử lý và quản lý tài nguyên.
* **Ưu điểm**:
  + **Quản lý tập trung**: Dễ dàng quản lý tài nguyên, dữ liệu, và bảo mật tại máy chủ. Ví dụ, cơ sở dữ liệu được lưu trữ và quản lý tại một máy chủ trung tâm.
  + **Tính đơn giản**: Thiết kế rõ ràng, với vai trò phân biệt giữa client và server, dễ triển khai và bảo trì.
  + **Khả năng mở rộng theo chiều dọc**: Có thể nâng cấp máy chủ (thêm CPU, RAM) để xử lý nhiều yêu cầu hơn.
  + **Bảo mật tốt hơn**: Máy chủ có thể kiểm soát truy cập và xác thực người dùng tập trung.
* **Nhược điểm**:
  + **Điểm thất bại đơn (Single Point of Failure)**: Nếu máy chủ gặp sự cố, toàn bộ hệ thống có thể ngừng hoạt động.
  + **Tắc nghẽn (Bottleneck)**: Máy chủ trung tâm có thể bị quá tải khi số lượng client tăng cao.
  + **Khó mở rộng theo chiều ngang**: Thêm nhiều máy chủ đòi hỏi cơ chế đồng bộ hóa phức tạp (như load balancing).
  + **Phụ thuộc vào máy chủ**: Client không thể hoạt động nếu mất kết nối với server.
* **Ví dụ**: Hệ thống web (client là trình duyệt, server là máy chủ web như Apache), email (client là Outlook, server là máy chủ email).

**b. Kiến trúc ngang hàng (Peer-to-Peer Architecture, P2P)**

* **Định nghĩa**: Trong kiến trúc ngang hàng, các nút (peers) có vai trò bình đẳng, vừa đóng vai trò là client vừa là server, chia sẻ tài nguyên trực tiếp với nhau mà không cần máy chủ trung tâm.
* **Ưu điểm**:
  + **Không có điểm thất bại đơn**: Vì không phụ thuộc vào máy chủ trung tâm, hệ thống vẫn hoạt động khi một số nút thất bại.
  + **Khả năng mở rộng tốt**: Hệ thống dễ dàng mở rộng khi thêm nút mới, vì mỗi nút đóng góp tài nguyên (CPU, băng thông, lưu trữ).
  + **Phân tải tự nhiên**: Tài nguyên được chia sẻ giữa các nút, giảm nguy cơ tắc nghẽn tại một điểm duy nhất.
  + **Chi phí thấp**: Không cần đầu tư vào máy chủ mạnh, tận dụng tài nguyên của các nút tham gia.
* **Nhược điểm**:
  + **Quản lý phức tạp**: Khó kiểm soát tài nguyên, bảo mật, và đồng bộ hóa dữ liệu do không có điểm quản lý trung tâm.
  + **Bảo mật yếu hơn**: Mỗi nút có thể là điểm tấn công, và khó áp dụng các chính sách bảo mật thống nhất.
  + **Hiệu suất không ổn định**: Hiệu năng phụ thuộc vào tài nguyên và trạng thái của các nút, có thể giảm khi nút rời đi hoặc hoạt động kém.
  + **Khó đảm bảo tính nhất quán**: Dữ liệu phân tán trên nhiều nút có thể dẫn đến xung đột hoặc không đồng bộ.
* **Ví dụ**: Hệ thống chia sẻ tệp BitTorrent, blockchain (như Bitcoin), hoặc mạng truyền thông như Skype (trong các phiên bản cũ).

**So sánh tổng quan**

| **Tiêu chí** | **Kiến trúc phân cấp** | **Kiến trúc ngang hàng** |
| --- | --- | --- |
| **Cấu trúc** | Máy khách và máy chủ phân vai rõ ràng | Các nút bình đẳng, vừa là client vừa là server |
| **Quản lý** | Tập trung, dễ kiểm soát | Phân tán, khó quản lý |
| **Khả năng chịu lỗi** | Có điểm thất bại đơn (máy chủ) | Không có điểm thất bại đơn |
| **Khả năng mở rộng** | Dễ mở rộng chiều dọc, khó chiều ngang | Dễ mở rộng chiều ngang |
| **Bảo mật** | Dễ kiểm soát tại máy chủ | Khó kiểm soát, dễ bị tấn công |
| **Hiệu suất** | Ổn định nếu máy chủ đủ mạnh | Không ổn định, phụ thuộc vào nút |
| **Ứng dụng** | Web, email, cơ sở dữ liệu | Chia sẻ tệp, blockchain, IoT |

**Nhận xét**: Kiến trúc phân cấp phù hợp với các hệ thống cần quản lý tập trung và bảo mật cao, nhưng dễ gặp vấn đề về tắc nghẽn và điểm thất bại đơn. Kiến trúc ngang hàng lý tưởng cho các ứng dụng cần mở rộng lớn và phân tán, nhưng phức tạp hơn trong quản lý và bảo mật.

**2. Trình bày bốn mô hình hệ thống phân tán và ví dụ ứng dụng điển hình**

Hệ thống phân tán có nhiều mô hình tổ chức khác nhau, mỗi mô hình phù hợp với một loại ứng dụng cụ thể. Dưới đây là bốn mô hình chính: **phân tầng**, **đối tượng phân tán**, **kênh sự kiện**, và **dữ liệu tập trung**, cùng với ví dụ ứng dụng:

**a. Mô hình phân tầng (Layered Model)**

* **Mô tả**: Hệ thống được tổ chức thành nhiều tầng (layers), mỗi tầng cung cấp một chức năng cụ thể và tương tác với các tầng khác theo cách có tổ chức. Các tầng thấp cung cấp dịch vụ cho các tầng cao hơn.
* **Đặc điểm**:
  + Tầng thấp nhất thường liên quan đến phần cứng hoặc mạng, tầng cao nhất tương tác với người dùng.
  + Tăng tính mô-đun, dễ bảo trì và phát triển.
* **Ví dụ ứng dụng**:
  + **Hệ thống web 3 tầng**: Tầng trình bày (presentation layer) là giao diện web (HTML/CSS), tầng logic nghiệp vụ (business logic layer) xử lý yêu cầu (máy chủ ứng dụng như Node.js), và tầng dữ liệu (data layer) lưu trữ thông tin (MySQL).
* **Ưu điểm**: Tách biệt chức năng, dễ mở rộng từng tầng.
* **Nhược điểm**: Có thể tăng độ trễ do giao tiếp giữa các tầng.

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

* **Mô tả**: Hệ thống được tổ chức thành các đối tượng (objects) phân tán trên nhiều nút, giao tiếp thông qua các lời gọi phương thức từ xa (remote method invocation, RMI) hoặc giao diện lập trình (API).
* **Đặc điểm**:
  + Các đối tượng đóng gói dữ liệu và hành vi, che giấu chi tiết triển khai.
  + Phù hợp với các ứng dụng hướng đối tượng.
* **Ví dụ ứng dụng**:
  + **Hệ thống CORBA hoặc Java RMI**: Một ứng dụng doanh nghiệp sử dụng CORBA để các đối tượng phân tán (như quản lý kho) giao tiếp giữa các máy chủ ở các chi nhánh khác nhau.
* **Ưu điểm**: Tăng tính tái sử dụng và tính đóng gói.
* **Nhược điểm**: Phức tạp trong triển khai và hiệu suất thấp hơn do lời gọi từ xa.

**c. Mô hình kênh sự kiện (Event-Based Model)**

* **Mô tả**: Các thành phần trong hệ thống giao tiếp thông qua các sự kiện (events), được gửi và nhận qua một kênh sự kiện (event bus). Các nút đăng ký (subscribe) để nhận sự kiện và phản ứng khi sự kiện xảy ra.
* **Đặc điểm**:
  + Giao tiếp bất đồng bộ, giảm sự phụ thuộc trực tiếp giữa các thành phần.
  + Phù hợp với hệ thống yêu cầu phản hồi nhanh và xử lý sự kiện lớn.
* **Ví dụ ứng dụng**:
  + **Apache Kafka**: Một hệ thống streaming dữ liệu sử dụng Kafka để truyền các sự kiện (như nhật ký người dùng) từ ứng dụng web đến các hệ thống phân tích.
* **Ưu điểm**: Linh hoạt, dễ mở rộng, hỗ trợ xử lý bất đồng bộ.
* **Nhược điểm**: Khó theo dõi luồng sự kiện và đảm bảo thứ tự.

**d. Mô hình dữ liệu tập trung (Data-Centric Model)**

* **Mô tả**: Hệ thống tập trung vào việc chia sẻ và quản lý dữ liệu thông qua một không gian dữ liệu chung (shared data space). Các nút đọc/ghi dữ liệu vào không gian này mà không cần giao tiếp trực tiếp.
* **Đặc điểm**:
  + Dữ liệu là trung tâm, các nút tương tác gián tiếp thông qua dữ liệu.
  + Phù hợp với các ứng dụng cần đồng bộ hóa dữ liệu lớn.
* **Ví dụ ứng dụng**:
  + **Hệ thống Linda hoặc Tuple Space**: Một hệ thống quản lý cảm biến IoT, nơi các cảm biến ghi dữ liệu (nhiệt độ, độ ẩm) vào không gian dữ liệu chung, và các ứng dụng đọc dữ liệu để phân tích.
* **Ưu điểm**: Giảm sự phụ thuộc giữa các nút, dễ đồng bộ hóa.
* **Nhược điểm**: Hiệu suất có thể giảm khi không gian dữ liệu lớn.

**Tóm tắt**

| **Mô hình** | **Mô tả** | **Ví dụ ứng dụng** |
| --- | --- | --- |
| **Phân tầng** | Tổ chức thành các tầng chức năng | Hệ thống web 3 tầng (HTML, Node.js, MySQL) |
| **Đối tượng phân tán** | Giao tiếp qua đối tượng từ xa | Ứng dụng doanh nghiệp với CORBA |
| **Kênh sự kiện** | Giao tiếp qua sự kiện bất đồng bộ | Streaming dữ liệu với Apache Kafka |
| **Dữ liệu tập trung** | Chia sẻ dữ liệu qua không gian chung | Hệ thống IoT với Linda/Tuple Space |

**3. Vai trò của phần mềm trung gian (middleware) trong kiến trúc khách chủ phân tán và ba tính năng chính**

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

* **Middleware** là lớp phần mềm nằm giữa ứng dụng (clients) và tài nguyên (servers) trong kiến trúc khách chủ phân tán, đóng vai trò cầu nối để che giấu sự phức tạp của hệ thống phân tán. Vai trò chính bao gồm:
  + **Tăng tính minh bạch**: Che giấu chi tiết về mạng, vị trí tài nguyên, hoặc lỗi, giúp client và server tương tác như trong một hệ thống cục bộ.
  + **Điều phối giao tiếp**: Quản lý việc truyền thông giữa client và server, đảm bảo yêu cầu được gửi và phản hồi được nhận đúng cách.
  + **Quản lý tài nguyên**: Phân bổ tài nguyên như CPU, bộ nhớ, hoặc băng thông, tối ưu hóa hiệu suất hệ thống.
  + **Hỗ trợ tích hợp**: Cho phép các ứng dụng và dịch vụ từ các nền tảng hoặc nhà cung cấp khác nhau làm việc cùng nhau.
  + **Ví dụ**: Trong một hệ thống web, middleware như Apache Tomcat xử lý các yêu cầu HTTP từ client, chuyển chúng đến ứng dụng Java trên server, và trả kết quả về client.

**Ba tính năng chính của Middleware**

1. **Giao tiếp từ xa (Remote Communication)**:
   * Cung cấp các cơ chế như RPC (Remote Procedure Call), RMI (Remote Method Invocation), hoặc messaging để client gọi hàm hoặc gửi yêu cầu đến server từ xa.
   * **Ví dụ**: Middleware như CORBA cho phép client gọi phương thức trên đối tượng nằm ở server khác.
2. **Quản lý tài nguyên và lập lịch (Resource Management and Scheduling)**:
   * Phân bổ tài nguyên (như CPU, bộ nhớ) và lập lịch các tác vụ để tối ưu hóa hiệu suất. Middleware có thể thực hiện load balancing để phân phối yêu cầu đều giữa các server.
   * **Ví dụ**: Middleware trong hệ thống đám mây như Kubernetes phân bổ tài nguyên cho các container dựa trên nhu cầu.
3. **Bảo mật và xác thực (Security and Authentication)**:
   * Đảm bảo an toàn cho giao tiếp giữa client và server thông qua xác thực (authentication), mã hóa (encryption), và phân quyền (authorization).
   * **Ví dụ**: Middleware như OAuth trong hệ thống web xác thực người dùng trước khi cho phép truy cập API.

**Câu 6:**

**1. Phân loại ba loại dịch vụ trong SOA kèm ví dụ điển hình**

**SOA (Service-Oriented Architecture)** là một mô hình kiến trúc trong đó các chức năng của hệ thống được tổ chức thành các dịch vụ độc lập, có thể tái sử dụng và tương tác với nhau qua các giao diện chuẩn hóa. Trong SOA, dịch vụ được phân loại thành ba loại chính: **cơ bản**, **tích hợp**, và **quy trình**. Dưới đây là phân tích chi tiết và ví dụ cho mỗi loại:

**a. Dịch vụ cơ bản (Basic Services)**

* **Định nghĩa**: Đây là các dịch vụ cấp thấp, cung cấp các chức năng đơn giản, độc lập, thường liên quan đến việc truy cập hoặc xử lý dữ liệu từ một nguồn cụ thể. Chúng là nền tảng cho các dịch vụ phức tạp hơn.
* **Đặc điểm**:
  + Tập trung vào một tác vụ cụ thể, không chứa logic nghiệp vụ phức tạp.
  + Thường tương tác trực tiếp với cơ sở dữ liệu hoặc hệ thống lưu trữ.
  + Có tính tái sử dụng cao trong nhiều ngữ cảnh.
* **Ví dụ điển hình**:
  + **Dịch vụ truy vấn thông tin khách hàng**: Một dịch vụ trả về thông tin chi tiết của khách hàng (tên, địa chỉ, số điện thoại) từ cơ sở dữ liệu dựa trên ID khách hàng. Ví dụ, API "GetCustomerInfo" trong hệ thống CRM.
* **Ứng dụng**: Dùng làm khối xây dựng cho các dịch vụ tích hợp hoặc quy trình.

**b. Dịch vụ tích hợp (Composite/Integration Services)**

* **Định nghĩa**: Các dịch vụ này kết hợp hoặc tổng hợp nhiều dịch vụ cơ bản hoặc các dịch vụ khác để thực hiện một chức năng phức tạp hơn. Chúng đóng vai trò trung gian, điều phối luồng dữ liệu giữa các dịch vụ.
* **Đặc điểm**:
  + Chứa logic nghiệp vụ để ghép nối hoặc xử lý dữ liệu từ nhiều nguồn.
  + Thường sử dụng các công cụ như ESB (Enterprise Service Bus) để tích hợp.
  + Đảm bảo tính tương tác giữa các hệ thống khác nhau.
* **Ví dụ điển hình**:
  + **Dịch vụ kiểm tra trạng thái đơn hàng**: Dịch vụ này gọi dịch vụ cơ bản "GetOrderDetails" (lấy chi tiết đơn hàng) và "GetShippingStatus" (lấy trạng thái giao hàng) để trả về thông tin tổng hợp về đơn hàng cho khách hàng.
* **Ứng dụng**: Tích hợp các hệ thống nội bộ (như CRM và ERP) hoặc kết nối với hệ thống bên ngoài.

**c. Dịch vụ quy trình (Process/Business Services)**

* **Định nghĩa**: Các dịch vụ cấp cao, đại diện cho một quy trình nghiệp vụ hoàn chỉnh, thường điều phối nhiều dịch vụ tích hợp và cơ bản để thực hiện một mục tiêu kinh doanh cụ thể.
* **Đặc điểm**:
  + Chứa logic nghiệp vụ phức tạp, mô hình hóa các quy trình kinh doanh.
  + Thường sử dụng các công cụ như BPM (Business Process Management) để quản lý luồng công việc.
  + Phụ thuộc vào các dịch vụ ở tầng dưới (tích hợp và cơ bản).
* **Ví dụ điển hình**:
  + **Dịch vụ xử lý đơn hàng**: Dịch vụ này thực hiện toàn bộ quy trình đặt hàng, từ kiểm tra kho (gọi dịch vụ cơ bản), thanh toán (gọi dịch vụ tích hợp với cổng thanh toán), đến thông báo giao hàng (gọi dịch vụ thông báo).
* **Ứng dụng**: Tự động hóa quy trình kinh doanh trong doanh nghiệp, như bán hàng, quản lý chuỗi cung ứng.

**Tóm tắt phân loại**

| **Loại dịch vụ** | **Mô tả** | **Ví dụ điển hình** |
| --- | --- | --- |
| **Cơ bản** | Chức năng đơn giản, truy cập dữ liệu | API lấy thông tin khách hàng (GetCustomerInfo) |
| **Tích hợp** | Tổng hợp nhiều dịch vụ cơ bản | Kiểm tra trạng thái đơn hàng |
| **Quy trình** | Mô hình hóa quy trình nghiệp vụ | Xử lý toàn bộ quy trình đặt hàng |

**2. Trình bày vòng đời của một dịch vụ SOA và những thách thức chính ở mỗi giai đoạn**

Vòng đời của một dịch vụ trong SOA bao gồm các giai đoạn từ phát triển đến vận hành sản xuất, với mục tiêu đảm bảo dịch vụ đáp ứng yêu cầu kinh doanh và hoạt động ổn định. Các giai đoạn chính bao gồm: **Thiết kế**, **Phát triển**, **Triển khai**, và **Vận hành sản xuất**. Dưới đây là phân tích chi tiết và thách thức ở mỗi giai đoạn:

**a. Giai đoạn thiết kế (Design)**

* **Mô tả**:
  + Xác định yêu cầu nghiệp vụ, phạm vi chức năng, và giao diện của dịch vụ.
  + Thiết kế cấu trúc dịch vụ, bao gồm giao thức (SOAP, REST), định dạng dữ liệu (XML, JSON), và các đặc tính như tính tái sử dụng, tính mở.
  + Đảm bảo dịch vụ tuân thủ các chuẩn SOA và tích hợp được với hệ thống hiện có.
* **Thách thức chính**:
  + **Xác định phạm vi dịch vụ**: Thiết kế dịch vụ quá lớn (chứa nhiều chức năng) hoặc quá nhỏ (thiếu tính tái sử dụng) có thể gây khó khăn trong việc sử dụng hoặc bảo trì.
  + **Đảm bảo tính mở và tương tác**: Lựa chọn chuẩn giao tiếp không phù hợp (như giao thức độc quyền) có thể hạn chế khả năng tích hợp với các hệ thống khác.
  + **Thiếu tài liệu hóa**: Nếu không tài liệu hóa rõ ràng giao diện và chức năng, nhà phát triển khác khó sử dụng dịch vụ.
* **Ví dụ**: Thiết kế API "GetCustomerInfo" với REST, trả về JSON, nhưng quên xác định các mã lỗi chuẩn (như 404, 500), dẫn đến khó xử lý lỗi ở phía client.

**b. Giai đoạn phát triển (Development)**

* **Mô tả**:
  + Lập trình dịch vụ dựa trên thiết kế, sử dụng các công cụ như Java, .NET, hoặc Python.
  + Tích hợp dịch vụ với các hệ thống cơ sở dữ liệu, ESB, hoặc các dịch vụ khác.
  + Kiểm thử đơn vị (unit testing) và kiểm thử tích hợp (integration testing) để đảm bảo dịch vụ hoạt động đúng.
* **Thách thức chính**:
  + **Tích hợp phức tạp**: Kết nối với các hệ thống cũ (legacy systems) hoặc các dịch vụ bên thứ ba có thể gặp vấn đề về giao thức hoặc định dạng dữ liệu.
  + **Hiệu suất**: Dịch vụ không được tối ưu hóa có thể gây độ trễ cao, đặc biệt khi xử lý lượng lớn yêu cầu.
  + **Quản lý phiên bản**: Thiếu cơ chế versioning (như v1, v2) có thể phá vỡ các ứng dụng client khi dịch vụ được cập nhật.
* **Ví dụ**: Phát triển dịch vụ "OrderProcessing" nhưng không kiểm thử tích hợp với cổng thanh toán, dẫn đến lỗi khi xử lý giao dịch thực tế.

**c. Giai đoạn triển khai (Deployment)**

* **Mô tả**:
  + Triển khai dịch vụ lên môi trường sản xuất hoặc môi trường thử nghiệm (staging).
  + Cấu hình các yếu tố như load balancing, bảo mật (OAuth, SSL), và giám sát (monitoring).
  + Đảm bảo dịch vụ sẵn sàng phục vụ các client (ứng dụng hoặc người dùng).
* **Thách thức chính**:
  + **Khả năng mở rộng**: Thiếu cấu hình load balancing hoặc cơ chế dự phòng có thể khiến dịch vụ bị quá tải khi lưu lượng tăng.
  + **Bảo mật**: Không mã hóa giao tiếp hoặc không xác thực đúng cách có thể dẫn đến các cuộc tấn công như man-in-the-middle.
  + **Downtime khi triển khai**: Việc triển khai phiên bản mới mà không có cơ chế blue-green deployment có thể gây gián đoạn dịch vụ.
* **Ví dụ**: Triển khai dịch vụ "GetShippingStatus" nhưng quên cấu hình SSL, khiến dữ liệu truyền đi bị lộ.

**d. Giai đoạn vận hành sản xuất (Production Operation)**

* **Mô tả**:
  + Quản lý, giám sát, và bảo trì dịch vụ trong môi trường sản xuất.
  + Thu thập log, giám sát hiệu suất, và xử lý lỗi hoặc sự cố.
  + Cập nhật dịch vụ để đáp ứng yêu cầu mới hoặc sửa lỗi.
* **Thách thức chính**:
  + **Giám sát và xử lý sự cố**: Thiếu công cụ giám sát (như Prometheus) hoặc log không đầy đủ có thể khiến khó phát hiện và khắc phục lỗi.
  + **Quản lý thay đổi**: Cập nhật dịch vụ mà không đảm bảo tương thích ngược (backward compatibility) có thể làm hỏng các client hiện có.
  + **Chi phí vận hành**: Dịch vụ tiêu tốn nhiều tài nguyên (như CPU, băng thông) có thể làm tăng chi phí vận hành, đặc biệt trong môi trường đám mây.
* **Ví dụ**: Dịch vụ "OrderProcessing" gặp lỗi do lưu lượng tăng đột biến trong Black Friday, nhưng không có cơ chế tự động mở rộng (auto-scaling), dẫn đến downtime.

**Tóm tắt vòng đời và thách thức**

| **Giai đoạn** | **Mô tả** | **Thách thức chính** |
| --- | --- | --- |
| **Thiết kế** | Xác định chức năng, giao diện dịch vụ | Phạm vi không rõ, thiếu chuẩn, tài liệu hóa |
| **Phát triển** | Lập trình, kiểm thử dịch vụ | Tích hợp phức tạp, hiệu suất, versioning |
| **Triển khai** | Đưa dịch vụ vào môi trường thực tế | Mở rộng, bảo mật, downtime |
| **Vận hành sản xuất** | Giám sát, bảo trì, cập nhật dịch vụ | Giám sát, quản lý thay đổi, chi phí |

**3. Kết luận**

* **Phân loại dịch vụ SOA**: Dịch vụ cơ bản cung cấp chức năng đơn giản, dịch vụ tích hợp tổng hợp nhiều dịch vụ, và dịch vụ quy trình mô hình hóa quy trình nghiệp vụ, mỗi loại phù hợp với các mục đích khác nhau.
* **Vòng đời dịch vụ SOA**: Bao gồm thiết kế, phát triển, triển khai, và vận hành, với các thách thức riêng ở mỗi giai đoạn, từ việc xác định phạm vi đến quản lý hiệu suất và bảo mật trong sản xuất.