# Bài tập Chương 11

## Bài tập 11.1

Các động lực chính cho sự ra đời của NoSQL bao gồm:

* Hạn chế của “one size fits all”: Hệ quản trị CSDL quan hệ (RDBMS) truyền thống cố gắng tích hợp hỗ trợ cho nhiều loại dữ liệu và ứng dụng, dẫn đến giảm hiệu năng và sự linh hoạt cho các ứng dụng có yêu cầu chuyên biệt.
* Khả năng mở rộng và tính sẵn sàng hạn chế: Kiến trúc 3 tầng truyền thống với một RDBMS ở tầng backend trở thành nút thắt cổ chai khi lượng truy cập tăng cao. Việc thêm máy chủ CSDL mới yêu cầu sao chép toàn bộ CSDL, rất tốn thời gian.
* Sự đánh đổi theo định lý CAP: Mặc dù thường bị hiểu sai, định lý CAP đã thúc đẩy việc xem xét lại sự đánh đổi giữa tính nhất quán mạnh (strong consistency) mà RDBMS cung cấp và tính sẵn sàng (availability) cũng như khả năng chịu phân hoạch mạng (partition tolerance), dẫn đến việc một số hệ thống NoSQL nới lỏng tính nhất quán để tăng khả năng mở rộng.

## Bài tập 11.2

* Tầm quan trọng của định lý CAP: Định lý CAP quan trọng vì nó hình thức hóa một sự đánh đổi cơ bản trong các hệ thống dữ liệu phân tán có sao chép: khi xảy ra phân hoạch mạng (P), hệ thống phải lựa chọn giữa tính nhất quán (C - tất cả các node thấy cùng một dữ liệu tại cùng một thời điểm) và tính sẵn sàng (A - mọi yêu cầu đều được trả lời).
* Áp dụng cho sao chép đa chủ (multimaster replication):
  1. Sao chép bất đồng bộ (Asynchronous replication): \*Các thuộc tính được duy trì: Availability (A) và Partition Tolerance (P).
     + Loại nhất quán đạt được: Nhất quán sau cùng (Eventual Consistency).
  2. Sao chép đồng bộ (Synchronous replication): \*Các thuộc tính được duy trì: Consistency (C) và Partition Tolerance (P).
     + Thuộc tính bị hy sinh: Availability (A) (trong phân vùng thiểu số - minority partition).

## Bài tập 11.3

| Loại NoSQL | Mô hình dữ liệu | Ngôn ngữ/Giao diện | Kiến trúc | Trường hợp sử dụng tốt nhất |
| --- | --- | --- | --- | --- |
| Key-Value | Các cặp key-value đơn giản | get/put/delete API | DHT, Băm nhất quán | Caching, quản lý session, hồ sơ người dùng đơn giản. |
| Document | Các tài liệu phức tạp, tự mô tả (JSON/BSON) | API phong phú, ngôn ngữ truy vấn dựa trên document | Sharding, Replica Sets | Quản lý nội dung, catalog sản phẩm, ứng dụng web linh hoạt. |
| Wide Column | Bảng, hàng được định danh bằng key, cột linh hoạt trong các họ cột (column families) | API, ngôn ngữ giống SQL (ví dụ: CQL) | Dựa trên Bigtable, Băm nhất quán | Dữ liệu chuỗi thời gian, logging, phân tích, ứng dụng ghi nhiều. |
| Graph | Nút (vertices), Cạnh (edges) và Thuộc tính (properties) | Ngôn ngữ duyệt đồ thị (ví dụ: Cypher, Gremlin) | Lưu trữ đồ thị gốc, thường là tập trung hoặc sao chép | Mạng xã hội, hệ thống gợi ý, phát hiện gian lận. |

## Bài tập 11.4

1. Thiết kế lược đồ cho các loại NoSQL: \*Key-Value: Cần nhiều loại key. Ví dụ: cust:<CID> trỏ đến đối tượng Customer, order:<OID> trỏ đến đối tượng Order. Các phép nối phải được thực hiện thủ công trong ứng dụng. Ưu điểm: Đơn giản. Nhược điểm: Logic ứng dụng phức tạp.
   * Document: Phù hợp tự nhiên. CUSTOMERS là một collection. ORDERS có thể là một collection riêng, hoặc tốt hơn là một mảng các document lồng nhau bên trong document CUSTOMER. PRODUCTS là một collection riêng. Ưu điểm: Ánh xạ tự nhiên, tránh nhiều phép nối. Nhược điểm: Document lồng sâu có thể lớn. \*Wide Column: Có thể có bảng CUSTOMERS với một họ cột cho thông tin cá nhân và một họ cột khác cho các đơn hàng. Ưu điểm: Lược đồ linh hoạt. Nhược điểm: Phi chuẩn hóa có thể dẫn đến dư thừa dữ liệu.
   * Graph: CUSTOMER, ORDER, PRODUCT là các node. PLACES (từ Customer đến Order), CONTAINS (từ Order đến Product) là các cạnh (relationships). Ưu điểm: Rất hiệu quả cho các truy vấn phức tạp về mối quan hệ. Nhược điểm: Có thể quá phức tạp cho các truy vấn tra cứu đơn hàng đơn giản.
2. Sản phẩm chứa các sản phẩm khác: \*Trong mô hình Graph, điều này được biểu diễn tự nhiên nhất bằng một cạnh đệ quy CONSISTS\_OF từ một node sản phẩm đến các node sản phẩm khác.
   * Trong mô hình Document, nó có thể là một mảng lồng nhau chứa các ID của sản phẩm thành phần.
   * Trong mô hình Key-Value, nó yêu cầu một key riêng để lưu danh sách các ID sản phẩm thành phần.

## Bài tập 11.5

* Tác động của phân mảnh đồ thị: Việc phân mảnh đồ thị tối ưu là bài toán NP-complete. Một phân mảnh kém sẽ tạo ra nhiều cạnh cắt ngang các node (inter-partition edges). Việc duyệt qua các cạnh này đòi hỏi giao tiếp mạng tốn kém, làm giảm hiệu năng và là nút thắt cổ chai chính cho việc mở rộng các CSDL đồ thị.
* Giải pháp:
  1. Không phân mảnh dữ liệu gốc (Master): Sử dụng kiến trúc sao chép toàn phần và cân bằng tải các truy vấn đọc. Ví dụ, Neo4j sử dụng causal clustering để mở rộng quy mô đọc, bằng cách định tuyến các truy vấn liên quan đến cùng một phần của đồ thị đến cùng một máy chủ đọc để tối đa hóa cache hit.
  2. Sao chép các đỉnh có bậc cao: Sao chép các “supernodes” trên nhiều phân vùng để các truy vấn bắt đầu từ chúng có thể thực hiện duyệt cục bộ.
  3. Sử dụng các thuật toán phân mảnh nhận biết đồ thị: Áp dụng các thuật toán heuristic phức tạp (như METIS) để giảm thiểu số cạnh cắt, mặc dù chi phí phân mảnh ban đầu cao.

## Bài tập 11.6

| Tiêu chí | F1 (NewSQL) | Parallel RDBMS (Tiêu chuẩn) |
| --- | --- | --- |
| Mô hình dữ liệu | Quan hệ, có mở rộng cho bảng phân cấp (hierarchical) và Protocol Buffers. | Quan hệ (chuẩn). |
| Ngôn ngữ truy vấn | Hỗ trợ đầy đủ SQL, có mở rộng. | Hỗ trợ đầy đủ SQL. |
| Tính nhất quán | Mạnh (ACID), được cung cấp bởi Spanner. | Mạnh (ACID). |
| Khả năng mở rộng | Được thiết kế cho quy mô toàn cầu (geo-scale) trên kiến trúc shared-nothing. | Mở rộng tốt trên cluster shared-nothing nhưng thường trong một trung tâm dữ liệu. |
| Tính sẵn sàng | Rất cao, tận dụng sao chép đồng bộ, đa trung tâm dữ liệu của Spanner. | Cao, thông qua các cơ chế sao chép và chuyển đổi dự phòng (failover). |

## Bài tập 11.7

* Điểm chung:
  + Cả hai đều cung cấp một giao diện thống nhất để truy vấn nhiều nguồn dữ liệu không đồng nhất.
  + Thường sử dụng kiến trúc mediator-wrapper.
* Điểm khác biệt:
  + Phạm vi: Các hệ thống tích hợp dữ liệu (Chương 7) chủ yếu tập trung vào việc tích hợp các CSDL có cấu trúc (thường là quan hệ). Polystores được thiết kế cho bối cảnh hiện đại, tích hợp nhiều loại data store hơn bao gồm NoSQL, HDFS, và RDBMS.
  + Lược đồ (Schema): Tích hợp dữ liệu truyền thống đặt nặng việc tạo ra một lược đồ toàn cục (GCS) thông qua GAV hoặc LAV. Nhiều polystore linh hoạt hơn, có thể không cần GCS tĩnh mà cho phép truy vấn trực tiếp bằng ngôn ngữ gốc hoặc định nghĩa lược đồ “on-the-fly”.
  + Độ khớp nối (Coupling): Polystores giới thiệu các khái niệm về tích hợp chặt chẽ (tightly coupled) và lai (hybrid), nơi polystore có thể kiểm soát trực tiếp lớp lưu trữ bên dưới, điều này ít phổ biến hơn trong các hệ thống tích hợp dữ liệu truyền thống.

## Bài tập 11.8

* Vấn đề: Vấn đề cốt lõi là sự không đồng nhất của các mô hình giao dịch. RDBMS hỗ trợ ACID và 2PC. Nhiều hệ thống NoSQL không hỗ trợ giao dịch đa mục, chỉ cung cấp tính nhất quán sau cùng, hoặc chỉ có tính nguyên tử cho một mục duy nhất. Không thể đảm bảo commit nguyên tử toàn cục nếu một thành phần tham gia thậm chí không thể đảm bảo trạng thái “prepare-to-commit”.
* Hướng giải pháp:
  1. Giới hạn ở mẫu số chung: Chỉ cho phép các giao dịch chạy trên tập hợp con các data store có hỗ trợ mô hình giao dịch tương thích (ví dụ: tất cả đều hỗ trợ 2PC).
  2. Giao dịch bù trừ (Compensating Transactions - Sagas): Đối với các data store không hỗ trợ 2PC, sử dụng phương pháp saga. Nếu một giao dịch cục bộ thất bại, một chuỗi các giao dịch bù trừ sẽ được thực thi để hoàn tác các tác động của các giao dịch đã thành công. Điều này cung cấp tính nguyên tử nhưng không có tính cô lập (isolation).
  3. Nới lỏng tính nhất quán: Từ bỏ ACID cho các giao dịch toàn cục. Cung cấp một đảm bảo yếu hơn như tính nhất quán sau cùng và để ứng dụng tự xử lý các trạng thái không nhất quán tạm thời.
  4. Middleware chuyên dụng: Xây dựng một lớp middleware giao dịch phức tạp để mô phỏng các thuộc tính giao dịch trên các hệ thống không hỗ trợ, có thể sử dụng log và logic phục hồi phức tạp.