 Giới thiệu khái niệm **Map/Reduce framework**.

 Chiến lược sử dụng Map/Reduce.

* **Vấn đề đặt ra: làm sao để chứa dữ liệu lớn với khối lượng ngày càng tăng, có thể không có cấu trúc, và có thể được lưu rải rác trên nhiều ổ cứng.**
* **Cần xử lý lượng dữ liệu này mà không muốn lo việc song song hóa, giao tiếp giữa tiến trình, hay các chi tiết kỹ thuật phức tạp.**
* **Giải pháp là ‘bring computation to data !’**
* Nếu dữ liệu mang tính giao dịch (transactional, như dữ liệu khách hàng, hóa đơn) → phù hợp với cơ sở dữ liệu truyền thống (DBMS, schema, table, index).
* Nếu cần quét dữ liệu lớn và xử lý đơn giản lặp đi lặp lại → dùng Map/Reduce sẽ hiệu quả hơn.

(vì **Đặc điểm dữ liệu mang tính giao dịch:**

* Nhiều bản ghi nhỏ (khách hàng, hóa đơn, đơn hàng...).
* Thường xuyên **thêm, sửa, xóa, truy vấn nhanh**.
* Đặc điểm của BDMS là Tối ưu cho **truy vấn ngắn, nhanh, chính xác**.

)

Giới thiệu chi tiết **framework Map/Reduce**.

* Dữ liệu được biểu diễn dưới dạng cặp key-value
* **Người dùng phải định nghĩa 2 hàm:**
  + **Mapper:** áp dụng lên dữ liệu, sinh ra các cặp key-value.
  + **Reducer:** áp dụng lên kết quả trung gian, đưa ra kết quả cuối.
* Trong hệ thống xử lý dữ liệu lớn (big data), Hadoop chịu trách nhiệm Xử lý toàn bộ logistics: phân phối, thực thi song song, gom kết quả.
* Hadoop tự động lo các công việc như chia nhỏ dữ liệu, phân phối dữ liệu cho các node, theo dõi tiến trình, xử lý lỗi, gom kết quả,...
* tổ chức mọi thứ nhịp nhàng giữa các mapper và reducer.

**Luồng xử lý:**

1. Người dùng viết hàm map → sinh ra key-value.
2. Hadoop thực thi map trên các phân vùng dữ liệu.
3. Hadoop shuffle & group kết quả theo key → tạo dữ liệu trung gian.
4. Hadoop phân phát dữ liệu này cho reducer.
5. Reducer chạy và trả về kết quả cuối.

* **Mapper** là thành phần đầu tiên trong quá trình MapReduce.
* Nó **nhận đầu vào là dữ liệu thô** (ví dụ: file văn bản, log, v.v.), sau đó **xử lý** và **chuyển đổi** dữ liệu đó thành các cặp key-value trung gian (key, value).
* Mỗi mapper xử lý một phần nhỏ của toàn bộ dữ liệu (chia nhỏ để xử lý song song).
* **Ví dụ:** Nếu đếm số lần xuất hiện của mỗi từ, mapper sẽ nhận từng dòng văn bản và phát ra các cặp (từ, 1).

**2. Reducer là gì?**

* **Reducer** là thành phần tiếp theo, nhận đầu vào là các cặp key-value đã được gom nhóm (tất cả các value có cùng key sẽ được đưa vào cùng một reducer).
* Nó **tổng hợp** hoặc **xử lý** các giá trị tương ứng với từng key để cho ra kết quả cuối cùng.
* **Ví dụ:** Trong bài toán đếm từ, reducer sẽ nhận tất cả các (từ, 1) do mapper phát ra, cộng lại để ra tổng số lần xuất hiện của từng từ.

**Tóm lại:**

* **Mapper:** Phân tích, chuyển đổi dữ liệu đầu vào thành các cặp (key, value) trung gian.
* **Reducer:** Nhận các cặp (key, value) đã nhóm lại theo key, thực hiện tổng hợp/chuyển đổi cuối cùng để cho ra kết quả.

Trình bày chi tiết ví dụ **đếm từ (word count)** – ví dụ kinh điển trong Map/Reduce

 **Bài toán đặt ra:**

* Đếm số lần xuất hiện của mỗi từ trong một tập dữ liệu lớn (ví dụ toàn bộ loạt phim Star Wars + fanfiction + blogs).
* Nếu dữ liệu ít (1 tập phim) → làm trên 1 máy tính được.
* Nếu dữ liệu khổng lồ → cần Map/Reduce để xử lý song song và phân tán.

 **Chiến lược với Map/Reduce:**

* **Thiết kế key-value đơn giản:**
  + **Key:** từ (word).
  + **Value:** số 1 (mỗi lần từ xuất hiện).
* Hadoop sẽ lo việc **shuffle & group** các từ giống nhau lại, nên reducer chỉ cần cộng tổng.

 **Pseudocode:**

* **Mapper:**
  + Đọc từng dòng văn bản.
  + Tách dòng thành các từ.
  + Emit (word, 1) cho mỗi từ.
* **Reducer:**
  + Nhận tất cả cặp (word, 1) từ Hadoop (đã group theo từ).
  + Cộng dồn số lần xuất hiện.
  + Xuất ra (word, total\_count).

 **Luồng xử lý minh họa:**

1. Dữ liệu gốc nằm rải rác trên nhiều ổ đĩa.
2. Các hàm Map chạy song song, biến dữ liệu thành (word, 1).
3. Hadoop **shuffle & sort**: gom tất cả cặp có cùng từ lại.
4. Hàm Reduce tính tổng số lần xuất hiện cho mỗi từ.
5. **Kết quả cuối:** danh sách từ cùng số lần xuất hiện.

 Giới thiệu thêm ví dụ về MapReduce.

 Làm rõ các **nguyên tắc thiết kế**, **chiến lược**, và **các đánh đổi (trade-offs)** khi dùng MapReduce.

 **Ôn lại framework:**

* Người dùng phải định nghĩa **key-value pair**, **mapper**, **reducer**.
* Hadoop lo song song hóa, shuffling, grouping.

 **Quy tắc về số lượng mappers & reducers:**

* Hadoop mặc định **1 mapper / data split** (phụ thuộc vào block size & config).
* **Reducers:** khuyến nghị **1 reducer / core** để đảm bảo chạy song song.
* Tuy nhiên, số tối ưu phụ thuộc vào **kích thước block**, **loại tác vụ**:
  + Ít reducer → ít file output hơn.
  + Quá nhiều reducer → task có thể chạy lâu.
  + Cần cân bằng.

 **Nguyên tắc chọn key-value:**

* Nên **đơn giản** để Hadoop dễ shuffle và regroup.
* Key-value tốt là khi dễ dàng thu được output đúng.

 **Nguyên tắc phân rã bài toán (task decomposition):**

* Nếu dữ liệu/bài toán chia nhỏ được thành các phần, và đầu ra chỉ cần **gom nhóm key-value** → MapReduce phù hợp.
* Ví dụ: Word Count → tách nhỏ dữ liệu, mapper tạo cặp (word,1), reducer cộng tổng.

Mở rộng ngoài Word Count với một ví dụ phức tạp hơn – phân tích dữ liệu **Twitter**.

Bối cảnh dữ liệu:

Mỗi tweet có nhiều trường: ngày, nội dung, vị trí, metadata…

Bài toán:

Đếm số lần xuất hiện của từ theo từng ngày (word count by day).

Đếm tổng số lần xuất hiện của từ trên toàn bộ dữ liệu (baseline frequencies).

Giải pháp:

Task 1: Key = (date, word), Value = 1 → Hadoop group theo cặp ngày + từ → reducer cộng tổng.

Task 2: Chính là word count bình thường (key = word, value = 1). Có thể tái sử dụng code word count trước đó.

Cách thực thi:

Có thể chạy hai job MapReduce riêng biệt từ dữ liệu gốc.

Hoặc cascade (chuỗi) MapReduce jobs:

Dùng output của Task 1 (đã được tổng hợp theo ngày) → làm input cho Task 2.

Ưu điểm: dữ liệu đầu vào nhỏ hơn nhiều so với toàn bộ dataset ban đầu.

Nguyên tắc rút ra:

Có thể kết hợp nhiều job MapReduce nối tiếp để giải quyết các bài toán nhiều bước.

Composite key (ví dụ: (date, word)) là chiến lược hữu ích để mở rộng.

Chuẩn bị cho video tiếp theo: Xem ví dụ về join dữ liệu, đòi hỏi đưa thêm thông tin vào value field