HỌC VIỆN CÔNG NGHỆ BƯU CHÍNH VIỄN THÔNG
CƠ SỞ THÀNH PHỐ HỒ CHÍ MINH
KHOA CÔNG NGHỆ THÔNG TIN II

BÁO CÁO CUỐI KÌ

HỌC PHẦN: CHUYÊN ĐỀ CÔNG NGHỆ PHẦN MỀM

CHUYÊN ĐỀ 4
Nghiên cứu cách xây dựng chỉ mục phân tán

Sinh viên thực hiện

Đỗ Tấn Kha – N19DCCN085

Cao Thanh Nhàn – N19DCCN125

Vũ Thị Hồng Oanh – N19DCCN135

T3 - 2023

Sơ lược MapReduce

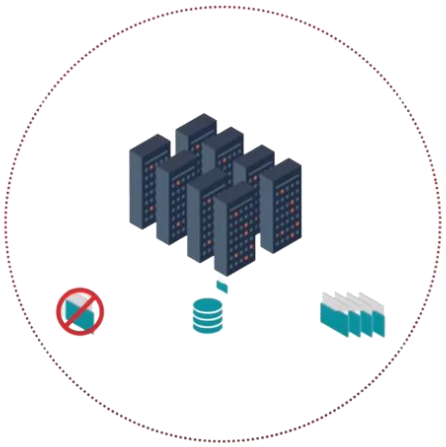
MapReduce là một kiến trúc tính toán phân tán được giới thiệu bởi Jeffrey Dean và Sanjay Ghemawat của Google vào năm 2004. Nó cung cấp một cách tiếp cận đơn giản và hiệu quả để xử lý dữ liệu lớn trên trên các cụm máy tính phân tán.

Người dùng định nghĩa một hàm Map xử lý một cặp key - value để tạo ra một tập các key - value trung gian và cũng định nghĩa một hàm Reduce để hợp nhất tất cả các value trung gian cùng kết nối đến cùng 1 key trung gian. Nhiều task trong thế giới thực có thể biểu diễn được với mô hình MapReduce này.

Các chương trình được theo kiểu hàm này được thực thi song song trên các cụm lớn các máy tính. Hệ thống run-time đảm nhận các chi tiết về phân vùng dữ liệu đầu vào, lên lịch thực hiện chương trình trên một nhóm máy, xử lý lỗi máy và quản lý giao tiếp giữa các máy được yêu cầu. Điều này cho phép các lập trình viên không có kinh nghiệm về hệ các hệ thống song song và phân tán dễ dàng sử dụng tài nguyên của 1 hệ thống phân tán lớn.

Lý do sử dụng MapReduce

Trước năm 2004, lượng dữ liệu khổng lồ được lưu trữ trong các server đơn. Nếu bất kỳ chương trình nào truy vấn dữ liệu lưu trữ trong nhiều server, việc tích hợp hợp lý các kết quả tìm kiếm và phân tích dữ liệu là một cơn ác mộng. Chưa kể những nổ lực và chi phí lớn liên quan. Nguy cơ mất dữ liệu, thách thức trong việc sao lưu dữ liệu và khả năng mở rộng giảm gây nên vấn đề khủng hoảng về sắp xếp.



Để giải quyết vấn đề này, Google đã giới thiệu đến MapReduce vào tháng 12 – 2004. Với MapReduce, việc phân tích tập dữ liệu được làm trong vòng chưa đầy 10 phút thay vì phải mất nhiều ngày. Những câu truy vấn có thể chạy đồng thời trên nhiều server, các kết quả tìm kiếm có thể tích hợp một cách hợp lý và dữ liệu có thể được phân tích trong thời gian thực.

Ưu điểm của MapReduce là khả năng chịu lỗi và khả năng mở rộng của nó. Lý do chính để thực hiện Mapping và Reducing là tăng tốc độ thực hiện công việc của một quy trình cụ thể. Điều này có thể được làm bằng cách chia một quy trình thành một số task, do đó cho phép xử lý song song.

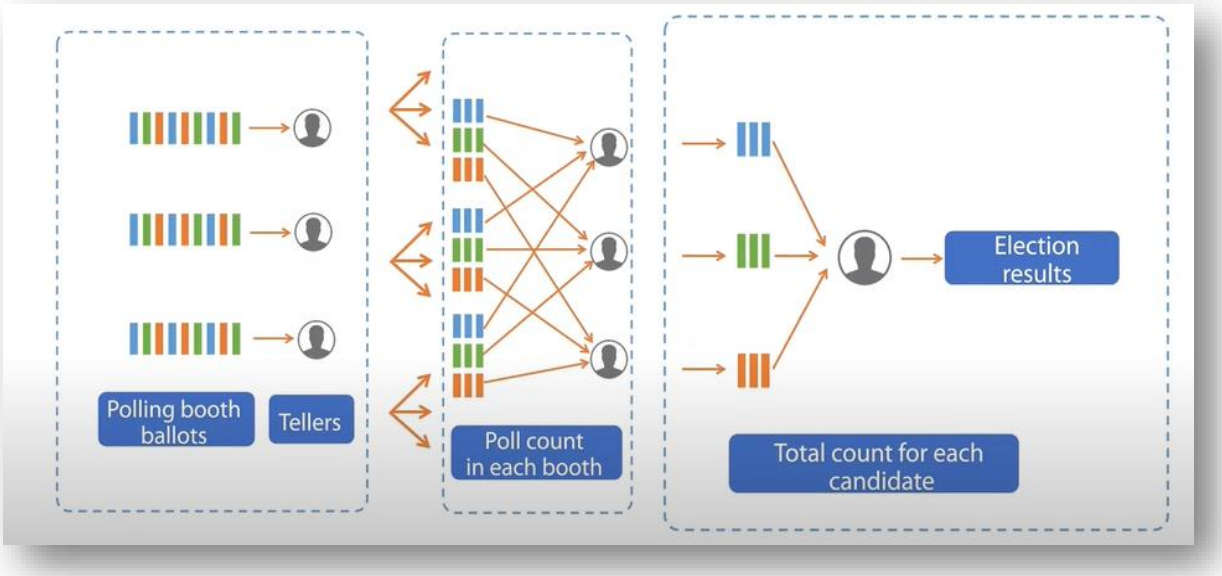
Mô hình lập trình MapReduce

MapReduce có 2 hàm chính là Map và Reduce, đây là 2 hàm đã được định nghĩa bởi người dùng và nó cũng chính là 2 giai đoạn chính trong quá trình xử lý dữ liệu của MapReduce. Nhiệm vụ cụ thể của từng hàm như sau:

Funtion	Input	Output
Map	(k1, v1)	List(k2, v2)
Reduce	(k2, list(v2))	(k2,v3)

Hàm Map lấy tham số đầu vào là 1 cặp key – value (k1, v1) và tạo ra 1 tập các cặp key – value list(k2, v2) gọi là các cặp key – value trung gian. Các value trung gian cùng liên quan đến 1 key trung gian được tổ hợp lại với nhau (k2, list(v2)) và truyền đến hàm Reduce.

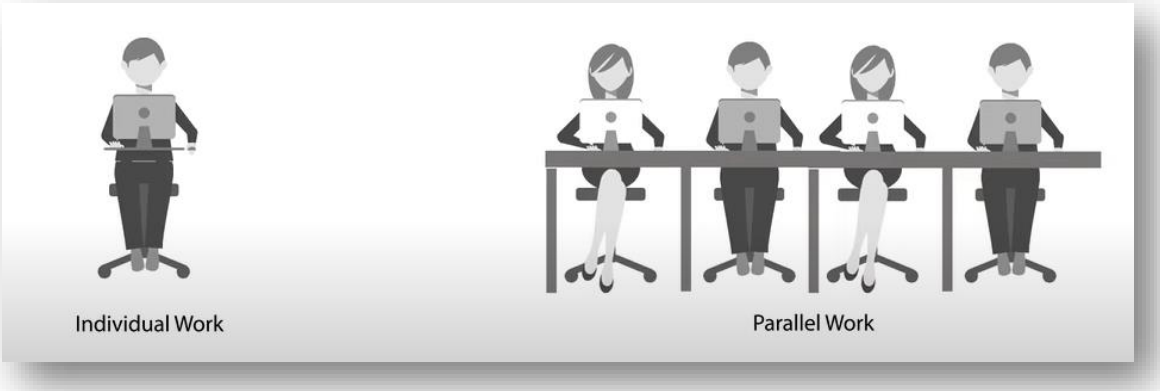
Hàm Reduce lấy tham số đầu vào là (k2, list(v2)), hợp nhất các giá trị list(v2) tạo thành một tập các giá trị có thể thể nhỏ hơn. Thông thường chỉ có 0 hoặc 1 giá trị đầu ra cho mỗi lần gọi Reduce.



Ví dụ với bài toán Đếm số phiếu bầu của 1 cuộc bầu cử.

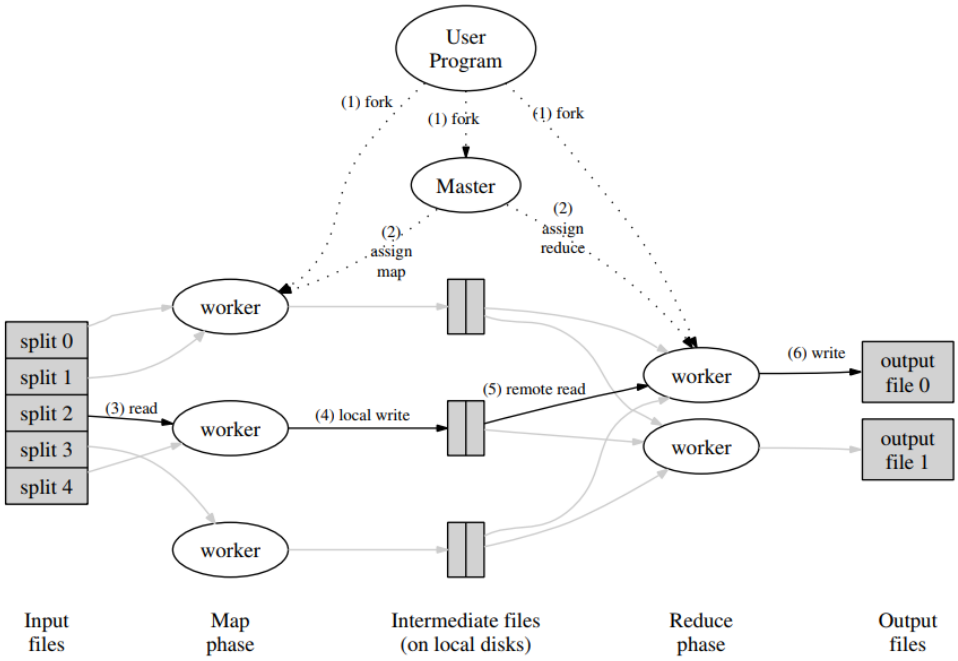
Sau khi kết thúc quá trình bỏ phiếu, các teller của tất cả các booth nhận các phiếu bầu và phân thành các nhóm phiếu theo từng ứng cử viên (3 màu), quá trình này được các teller thực hiện song song như là 1 single job (mỗi teller chỉ kiểm phiếu bầu của 1 ứng cử viên). Đây được xem như là pha Map với đầu vào hàm Map là (key=boothID, value = ballots_bID) và đầu ra là (key = candidateID, value = ballot_of_cID). Các ballot_of_cID của cùng 1 candidateID được tổ hợp lại với nhau thành (key = candidateID, value = list(ballot_of_cID))

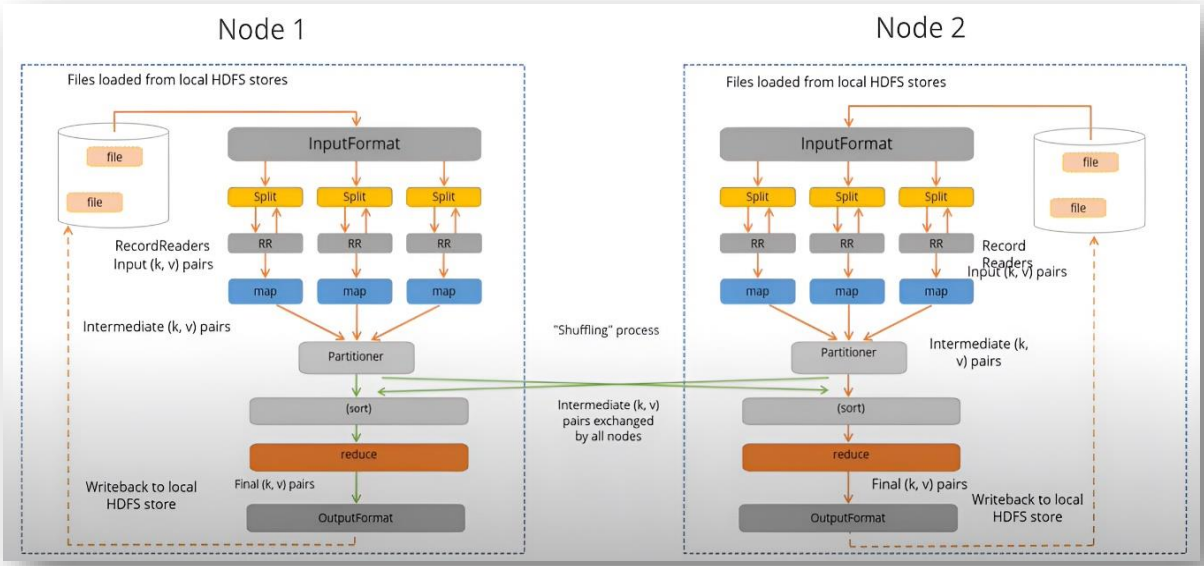
Cuối cùng các teller tính tổng số phiếu bầu cho mỗi ứng cử viên. Đây được xem như là pha Reduce với đầu vào là (key = candidateID, value = sum(list(ballot_of_cID))) và thu được kết quả bầu cử.



Nếu việc đếm phiếu bầu chỉ được thực hiện bởi 1 người có thể mất 1 tháng để nhận được kết quả bầu cử thì với việc nhiều người cùng thực hiện song song thì kết quả bầu cử thu được có thể chỉ trong 1 đến 2 ngày.

Các pha MapReduce





Phase	Description
Split	Đầu vào được phân tích cú pháp thành các bản ghi dưới dạng các cặp key-value. (key – value này là đầu vào của Map)
Map	Áp dụng Map function cho từng bản ghi để trả về 0, hoặc nhiều bảng ghi mới. Các output trung gian này được lưu trữ trong hệ thống file local như một file. Chúng được sắp xếp theo bucket number và sau đó là key. Thông báo Master Node về sự hoàn thành.
Partition	Mỗi mapper phải xác định reducer nào sẽ nhận từng output nào. Với bất kì giá trị key, bất kể Mapper nào sinh ra nó đều có phân vùng đích là giống nhau. Vì vậy đối với 1 từ duy nhất từ đó sẽ luôn đi đến cùng 1 phân vùng đích. Số lượng partition = số lượng reducer
Shuffle	Nạp dữ liệu ouput từ tất cả các map tasks tương ứng với reduce tasks bucket.
Sort	Sắp xếp tất cả các dữ liệu được nạp thành 1 single run
Reduce	Áp dụng hàm reduce tự định nghĩa đến merged run. Các đối số là 1 khóa và iter các giá trị tương ứng. Output được ghi vào 1 file.

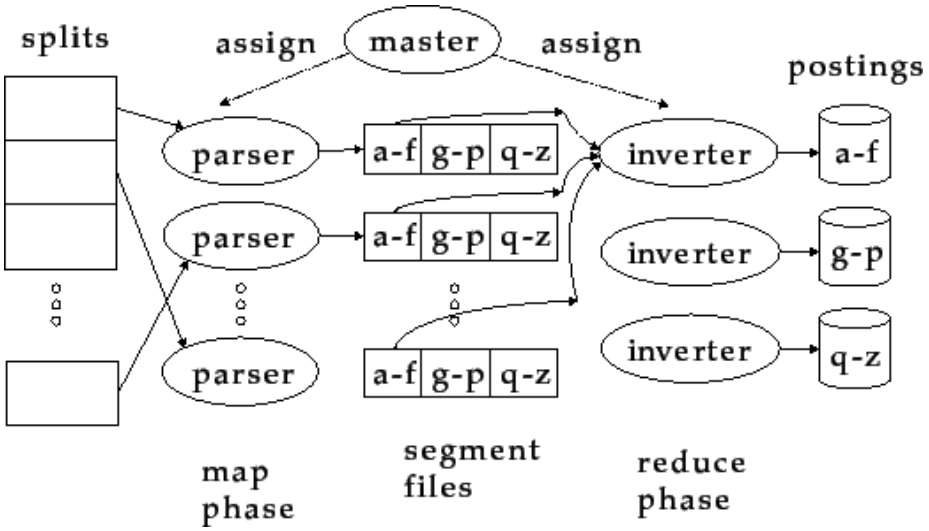
Xây dựng chỉ mục phân tán với MapReduce

Pseudo-code cho hàm Map và Reduce:

Map	Reduce

map(String key, String value):
// key: document name
// value: document contents
for each word w in value:
EmitIntermediate(w, key);
	reduce(String key, Iterator values):
	// key: a word
	// values: a sorted iter of postings;
	result = []
	For each value v in values:
	if v not in result:
	result.append(v);
	Emit(key, result);

Kiến trúc chỉ mục phân tán với số lượng partition = 3:



***Split Phase**:* Đầu vào các documents được phân tích thành các cặp key – value có dạng (docID, document). Ví dụ ta có tập gồm 2 tài liệu:

Document 1: “the quick brown fox jumps over the lazy dog”

Document 2: “the brown dog jumps over the brown fox”

Output: (1, “the quick brown fox jumps over the lazy dog”)

(2, “the brown dog jumps over the brown fox”)

Map Phase: Các mapper (parser) áp dụng hàm map được gán bởi Master Node cho mỗi cặp (docID, document) để trả về tập các key – value trung gian có dạng (term, docID).

Output: (the, 1), (quick, 1), (brown, 1), (fox, 1), (jumps, 1), (over, 1), (the, 1), (lazy, 1), (dog, 1)
(the, 2), (brown, 2), (dog, 2), (jumps, 2), (over, 2), (the, 2), (brown, 2), (fox, 2)

Các cặp key-value trung gian này được lưu trên file và sau đó thông báo sự hoàn thành về Master Node.

Partition Phase: Ta có số lượng partition = 3 (a - f, g – p, q - z) = số lượng reducer. Partitioner xác định Reducer nào sẽ nhận dữ liệu nào từ map tasks.

Shuffle Phase: Nạp dữ liệu output từ map tasks có phân vùng tương ứng với Reducer (inverter) tasks được thực hiện bởi master Node .

a – f: (brown, 1), (fox, 1), (dog, 1) / (brown, 2), (dog, 2), (brown, 2), (fox, 2)

g – p: (jumps, 1), (lazy, 1) / (jumps, 2), (over, 2)

q – z: (the, 1), (quick, 1), (the, 1) / (the, 2), (the, 2)

Sort Phase: Merge sort dữ liệu thành 1 luồng chạy đơn

a – f: (brown, 1), (brown, 2), (brown, 2), (dog, 1), (dog, 2), (fox, 1), (fox, 2)

g – p: (jumps, 1), (jumps, 2), (lazy, 1), (over, 2)

q – z: (quick, 1), (the, 1), (the, 1), (the, 2), (the, 2)

Reduce Phase: Áp dụng hàm reduce lên mỗi key trung gian và iter các value trung gian của nó và ghi output vào file.

a – f: (brown, [1, 2]), (dog, [1, 2]), (fox, [1, 2])

g – p: (jumps, [1, 2]), (lazy, [1]), (over, [2])

q – z: (quick, [1]), (the, [1, 2])