

25 YEARS ANNIVERSARY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY



HA NOI UNIVERSITY OF SCIENCE AND TECHNOLOGY
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Chương 6

Xử lý hàng loạt - phần 1

Bản đồ Giảm

Xử lý dữ liệu: MapReduce

- Khung MapReduce là dữ liệu mặc định của Hadoop công cụ xử lý •

MapReduce là một mô hình lập trình để xử lý dữ liệu

- nó không phải là một ngôn ngữ, một phong cách xử lý dữ liệu được tạo ra bởi Google

- Vẻ đẹp của MapReduce

- Sự đơn giản
- Tính linh hoạt
- Khả năng mở rộng

một công việc MR = {Nhiệm vụ riêng biệt}n

- MapReduce chia khói lượng công việc thành nhiều tác vụ độc lập và lên lịch chúng trên các nút cụm
- Công việc được thực hiện bởi mỗi tác vụ được thực hiện riêng biệt với nhau vì lý do khả năng mở rộng • Chi phí truyền thông cần thiết để giữ cho dữ liệu trên các nút được đồng bộ hóa mọi lúc sẽ ngăn cản mô hình thực hiện đáng tin cậy và hiệu quả ở quy mô lớn

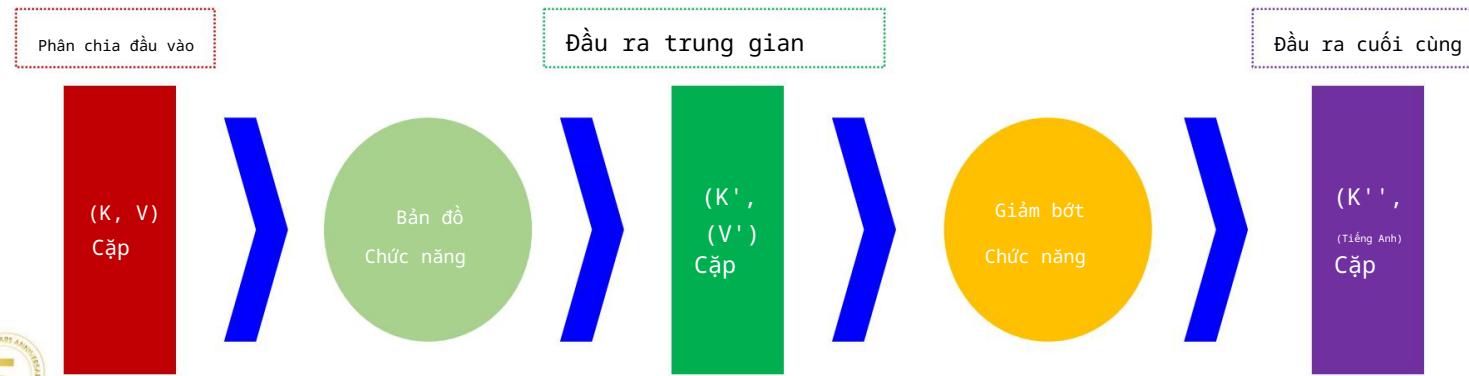
Phân phối dữ liệu

- Trong cụm MapReduce, dữ liệu thường được quản lý bởi hệ thống tệp phân tán (ví dụ: HDFS)
 - **Di chuyển mã đến dữ liệu chứ không phải dữ liệu đến mã**



Khóa và giá trị

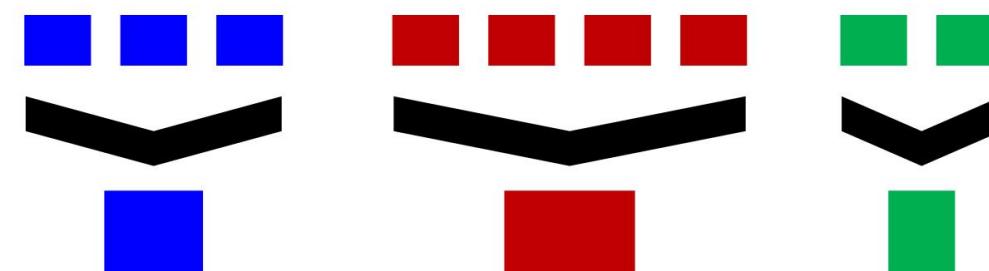
- Người lập trình trong MapReduce phải chỉ định hai các hàm, hàm map và hàm reduce thực hiện Mapper và Reducer trong một Chương trình MapReduce
- Trong MapReduce, các phần tử dữ liệu luôn được cấu trúc BẰNG cặp khóa-giá trị (tức là (K, V))
- Các hàm map và reduce nhận và phát ra (K, V) cặp



Phân vùng

- § Một tập hợp con khác của không gian khóa trung gian là được giao cho mỗi Reducer
- § Các tập hợp con này được gọi là phân vùng

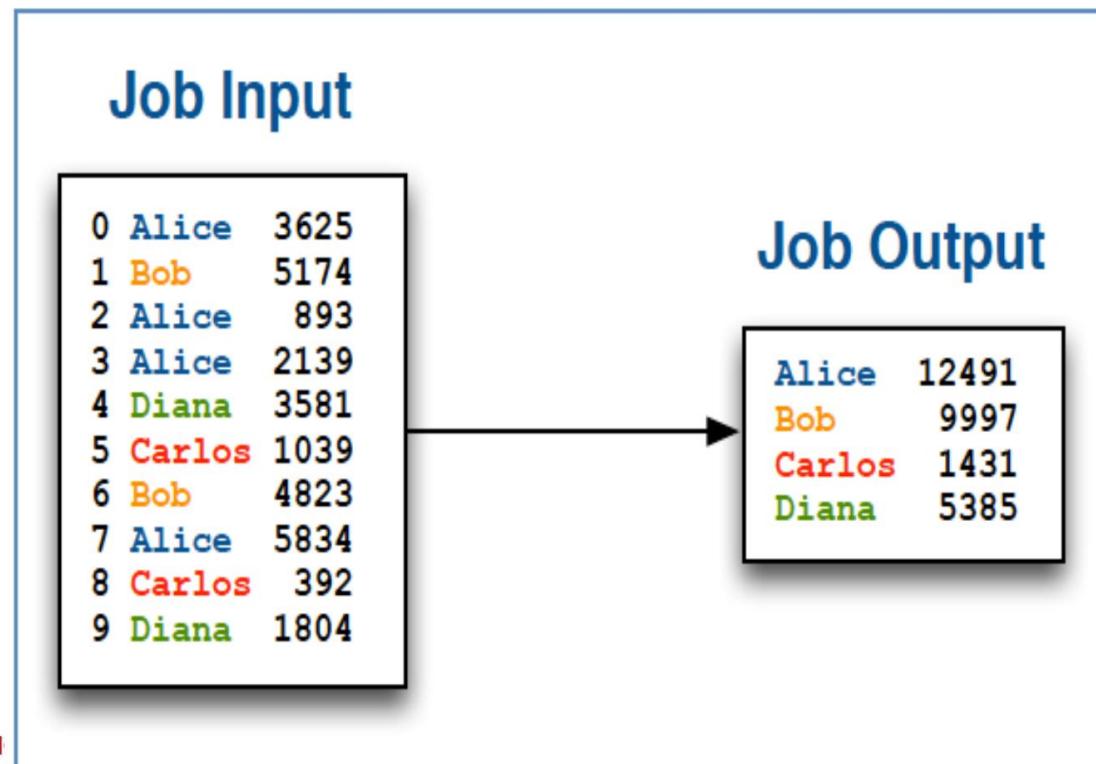
Các màu khác nhau đại diện
cho các phím khác nhau (có khả
năng) từ các Mapper khác nhau



Phân vùng là đầu vào cho Reducers

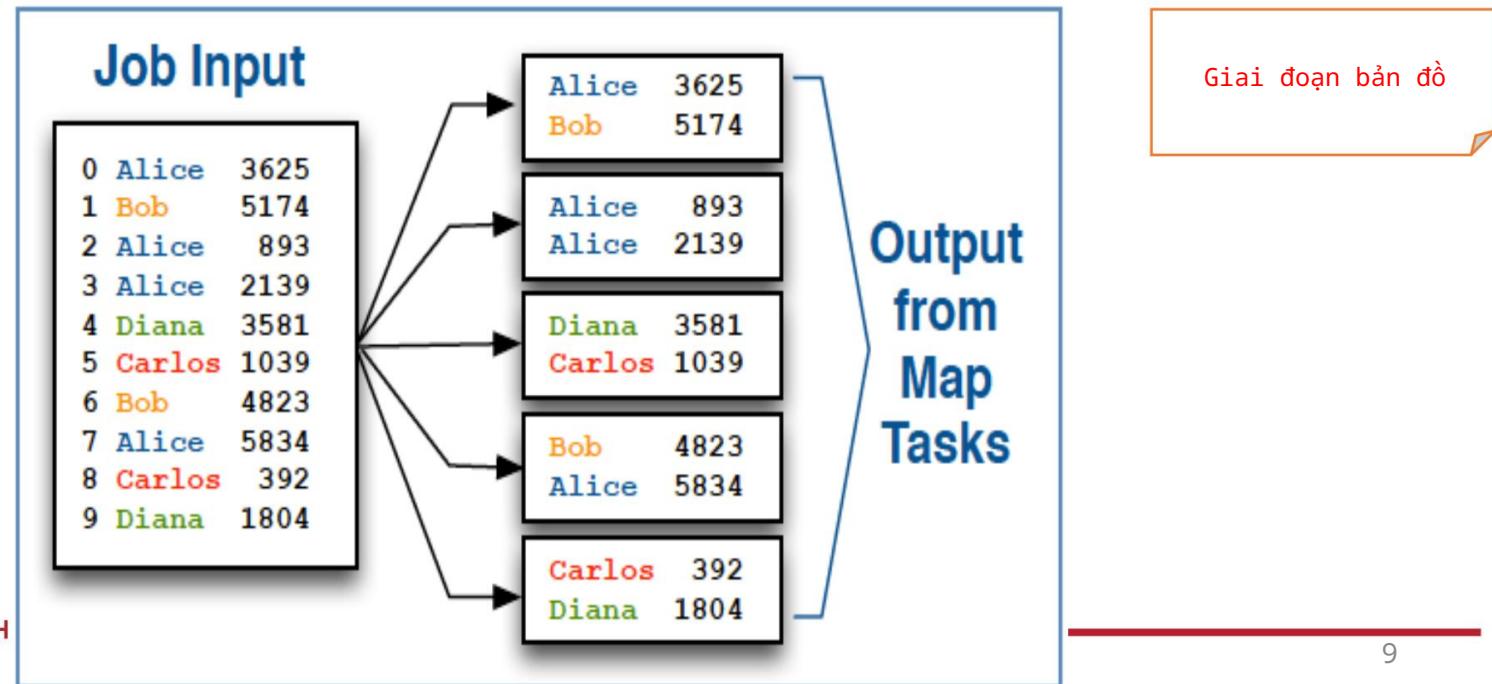
Ví dụ MapReduce

- Đầu vào: tệp văn bản chứa ID đơn hàng, tên nhân viên và số tiền bán hàng
- Đầu ra: tổng doanh số bán hàng của mỗi nhân viên



Giai đoạn bản đồ

- Hadoop chia công việc thành nhiều nhiệm vụ bản đồ riêng lẻ
 - Số lượng tác vụ bản đồ được xác định bởi lượng dữ liệu đầu vào
 - Mỗi tác vụ bản đồ nhận một phần dữ liệu đầu vào của toàn bộ công việc để xử lý
 - Người lập bản đồ xử lý một bản ghi đầu vào tại một thời điểm
 - Đối với mỗi bản ghi đầu vào, họ phát ra không hoặc nhiều bản ghi dưới dạng đầu ra
- Trong trường hợp này, tác vụ bản đồ chỉ đơn giản là phân tích cú pháp bản ghi đầu vào
 - Và sau đó phát ra các trường tên và giá cho mỗi trường như đầu ra



Trộn và sắp xếp

- Hadoop tự động sắp xếp và hợp nhất đầu ra từ tất cả nhiệm vụ bản đồ
 - Quá trình trung gian này được gọi là **xáo trộn và sắp xếp**
 - Kết quả được cung cấp để giảm nhiệm vụ

Map Task #1 Output

Alice	3625
Bob	5174

Map Task #2 Output

Alice	893
Alice	2139

Map Task #3 Output

Diana	3581
Carlos	1039

Map Task #4 Output

Bob	4823
Alice	5834

Map Task #5 Output

Carlos	392
Diana	1804

Input to Reduce Task #1

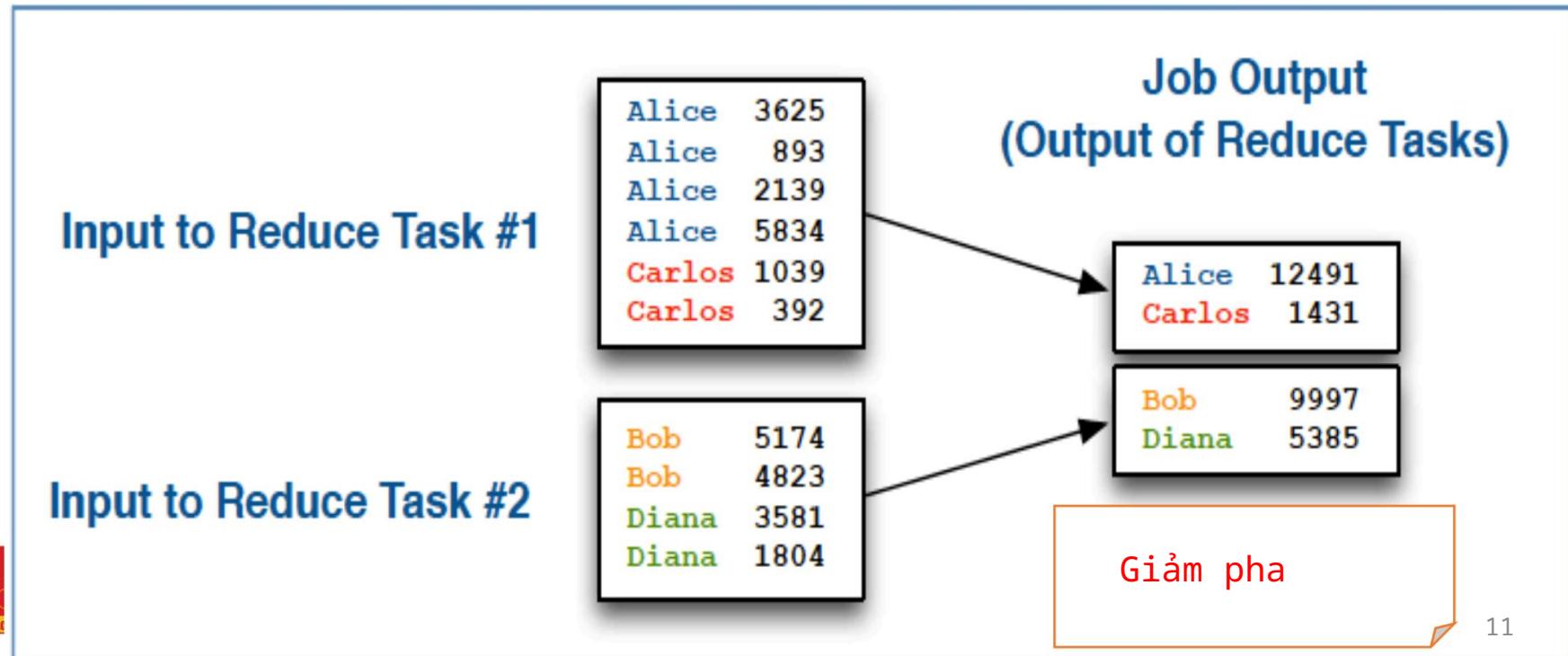
Alice	3625
Alice	893
Alice	2139
Alice	5834
Carlos	1039
Carlos	392

Bob	5174
Bob	4823
Diana	3581
Diana	1804

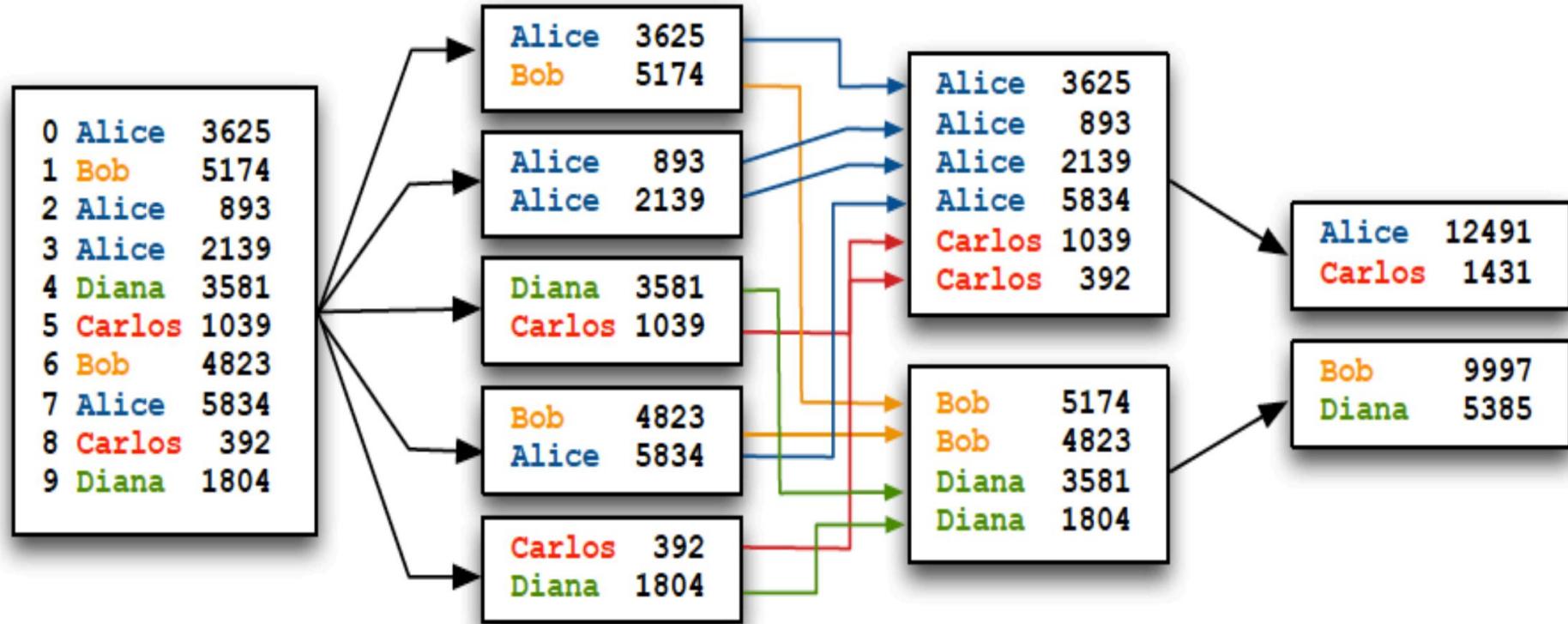
Giai đoạn xáo trộn và sắp xếp

Giảm pha

- Đầu vào bộ giảm tốc đến từ quá trình xáo trộn và sắp xếp
 - Giống như với map, hàm reduce nhận một bản ghi tại một thời điểm • Một reducer nhất định nhận tất cả các bản ghi cho một khóa nhất định
 - Đối với mỗi bản ghi đầu vào, reduce có thể phát ra không hoặc nhiều bản ghi đầu ra
- Hàm giảm của chúng tôi chỉ đơn giản là tính tổng cho mỗi người
 - Và phát ra tên nhân viên (khóa) và tổng số (giá trị) làm đầu ra

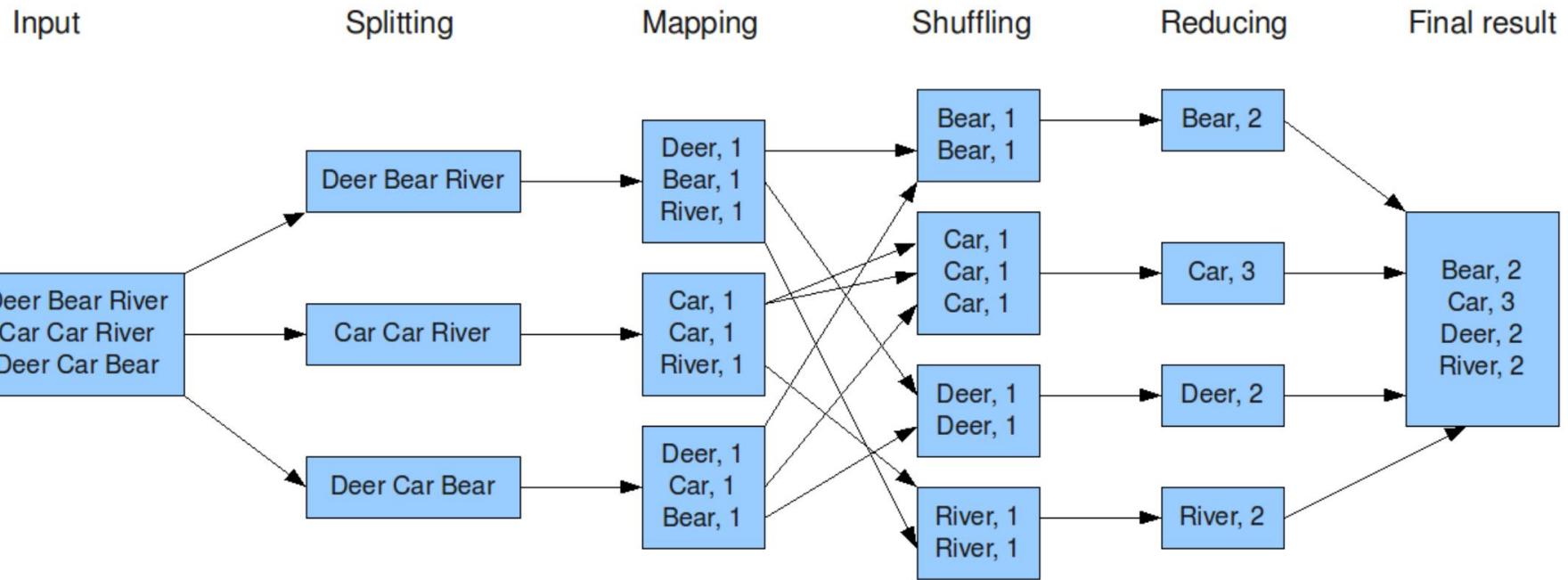


Luồng dữ liệu cho toàn bộ công việc MapReduce

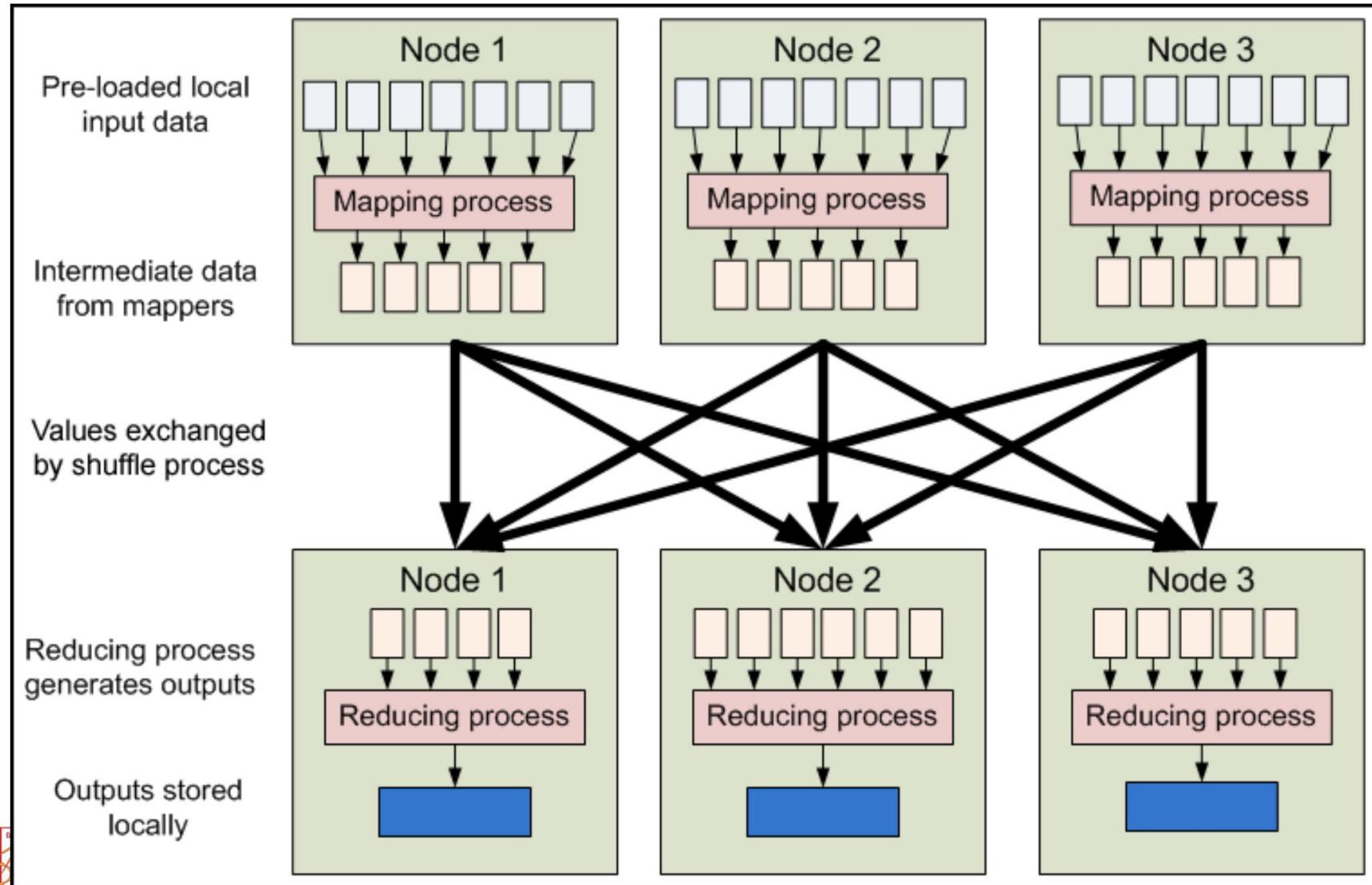


Luồng dữ liệu đếm từ

The overall MapReduce word count process



MapReduce - Luồng dữ liệu



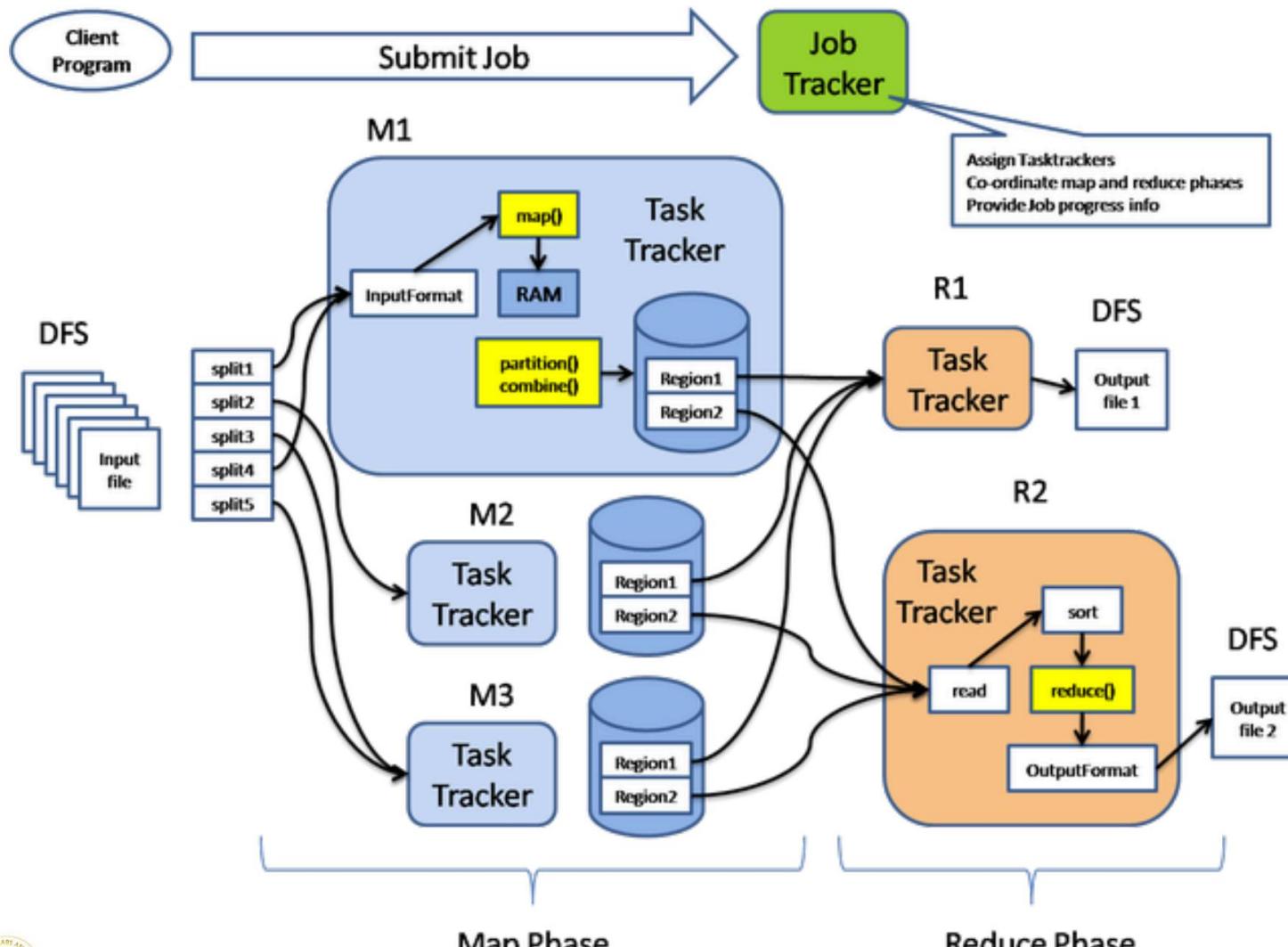
Ví dụ: Số từ (1)

```
9 import org.apache.hadoop.mapreduce.Job;
10 import org.apache.hadoop.mapreduce.Mapper;
11 import org.apache.hadoop.mapreduce.Reducer;
12 import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
13 import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
14 import org.apache.hadoop.util.GenericOptionsParser;
15
16
17
18
19 public class WordCount {
20 public static void main(String [] args) throws Exception
21 {
22 Configuration c=new Configuration();
23 String[] files=new GenericOptionsParser(c,args).getRemainingArgs();
24 Path input=new Path(files[0]);
25 Path output=new Path(files[1]);
26 Job j=new Job(c,"wordcount");
27 j.setJarByClass(WordCount.class);
28 j.setMapperClass(MapForWordCount.class);
29 j.setReducerClass(ReduceForWordCount.class);
30 j.setOutputKeyClass(Text.class);
31 j.setOutputValueClass(IntWritable.class);
32 FileInputFormat.addInputPath(j, input);
33 FileOutputFormat.setOutputPath(j, output);
34 System.exit(j.waitForCompletion(true)?0:1);
35 }
```

Ví dụ: Số từ (2)

```
36 public static class MapForWordCount extends Mapper<LongWritable, Text, Text, IntWritable>{
37     public void map(LongWritable key, Text value, Context con) throws IOException, InterruptedException
38     {
39         String line = value.toString();
40         String[] words = line.split(",");
41         for(String word: words )
42         {
43             Text outputKey = new Text(word.toUpperCase().trim());
44             IntWritable outputValue = new IntWritable(1);
45             con.write(outputKey, outputValue);
46         }
47     }
48 }
49
50 public static class ReduceForWordCount extends Reducer<Text, IntWritable, Text, IntWritable>
51 {
52     public void reduce(Text word, Iterable<IntWritable> values, Context con) throws IOException, InterruptedException
53     {
54         int sum = 0;
55         for(IntWritable value : values)
56         {
57             sum += value.get();
58         }
59         con.write(word, new IntWritable(sum));
60     }
}
```

Bản đồ giảm vòng đời



Thuật toán MapReduce

(C) <https://courses.cs.washington.edu/courses/cse490h/08au/lectures.htm>

Thuật toán cho MapReduce

- Sắp xếp
- Tìm kiếm
- TF-IDF
- BFS
- PageRank
- Các thuật toán tiên tiến hơn

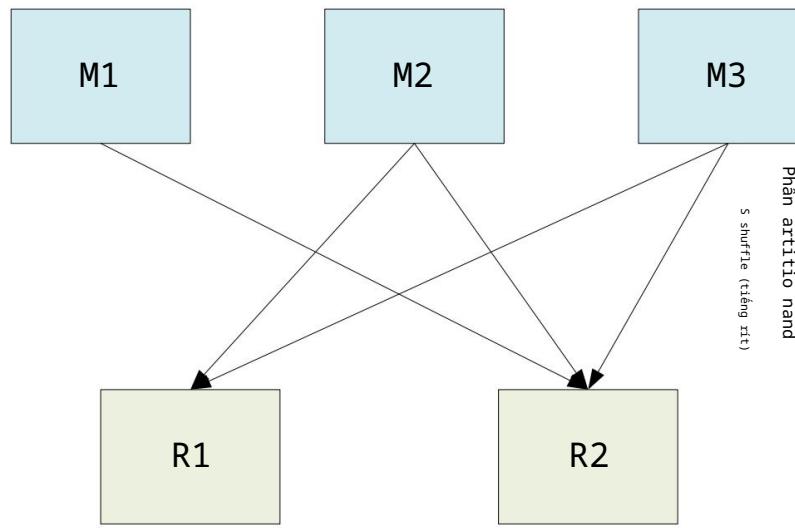
Thuật toán sắp xếp

- Được sử dụng để kiểm tra tốc độ thô của Hadoop
- Về cơ bản là “cuộc đua kéo IO”
- Đầu vào • Một tập hợp các tệp, một giá trị trên mỗi dòng • Khóa ánh xạ là tên tệp, số dòng • Giá trị ánh xạ là nội dung của dòng

Ý tưởng

- Tận dụng các thuộc tính của bộ giảm tốc: các cặp (khóa, giá trị) được xử lý theo thứ tự khóa; bản thân các bộ giảm tốc cũng được sắp xếp
- Mapper: Hàm nhận dạng cho giá trị
 $(k, v) \rightarrow (v, _)$
- Bộ giảm: Hàm đồng nhất $(k', _) \rightarrow (k', "")$

Ý tưởng (2)



- Các cặp (khóa, giá trị) từ trình ánh xạ được gửi đến một trình giảm cụ thể dựa trên hàm băm (khóa) •

Phải chọn hàm băm cho dữ liệu của bạn sao cho $k_1 < k_2 \Rightarrow$ hàm băm (k_1) < hàm băm (k_2)

Thuật toán tìm kiếm

- Đầu vào • Một tập hợp các tệp chứa các dòng văn bản • Một mẫu tìm kiếm để tìm • Khóa ánh xạ là tên tệp, số dòng • Giá trị ánh xạ là nội dung của dòng • Mẫu tìm kiếm được gửi dưới dạng tham số đặc biệt

Thuật toán tìm kiếm

- Mapper •

Cho (tên tệp, một số văn bản) và “mẫu”, nếu “văn bản” khớp với đầu ra “mẫu” (tên tệp, _)

- Reducer •

Hàm nhận dạng

Tối ưu hóa

- Khi một tập tin được tìm thấy là thú vị, chúng ta chỉ cần đánh dấu nó theo cách đó một lần
- Sử dụng hàm Combiner để gộp các cặp trùng lặp (tên tệp, _) thành một cặp duy nhất
 - Giảm I/O mạng

Thuật toán TF-IDF

- Tần suất thuật ngữ - Tần suất tài liệu ngược
 - Có liên quan đến xử lý văn bản
 - Thuật toán phân tích web phổ biến

$$tf_i = \frac{n_i}{\sum_k n_k}$$

$$idf_i = \log \frac{|D|}{|\{d : t_i \in d\}|}$$

$$tfidf = tf \cdot idf$$

- $|D|$: tổng số tài liệu trong kho dữ liệu
- $|\{d : t_i \in d\}|$: số lượng tài liệu có thuật ngữ t_i xuất hiện (tức là $n_i \neq 0$).

Sự quan sát

- Thông tin cần thiết
 - Số lần thuật ngữ X xuất hiện trong một tài liệu nhất định
 - Số lượng thuật ngữ trong mỗi tài liệu
 - Số lượng tài liệu X xuất hiện trong tổng số tài liệu

Công việc 1: Tần suất từ trong mỗi tài liệu

- Mapper •

Đầu vào: (docname, contents) •

Đầu ra: ((word, docname), 1)

- Bộ giảm tốc

• Tổng số đếm cho từ trong tài liệu • Đầu ra

((từ, tên tài liệu), n) • Bộ kết hợp

giống như Bộ thu gọn

Công việc 2: Đếm số từ trong tài liệu

- Mapper •

Đầu vào: ((word, docname), n)

- Đầu ra: (docname, (word, n))

- Bộ giảm tốc

- Tổng tần suất của các số n riêng lẻ trong cùng một tài liệu
 - Đưa dữ liệu gốc qua

- Đầu ra ((word, docname), (n, N))
 - = !

Công việc 3: Tân suất từ trong ngữ liệu

- Mapper •

Đầu vào: ((word, docname), (n, N))

- Đầu ra: (word, (docname, n, N, 1))

- Bộ giảm tốc

- Số lượng tài liệu có xuất hiện thuật ngữ từ d • Đầu ra ((từ, tên tài liệu), (n, N, d))

Công việc 4: Tính toán TF-IDF

- Mapper •

Đầu vào: ((word, docname), (n, N, d))

- Giả sử D đã biết (hoặc MR dễ tìm thấy) • Đầu ra ((word, docname), TF*IDF)

- Bộ giảm tốc

- Chỉ có chức năng nhận dạng

Suy nghĩ cuối cùng về TF-IDF

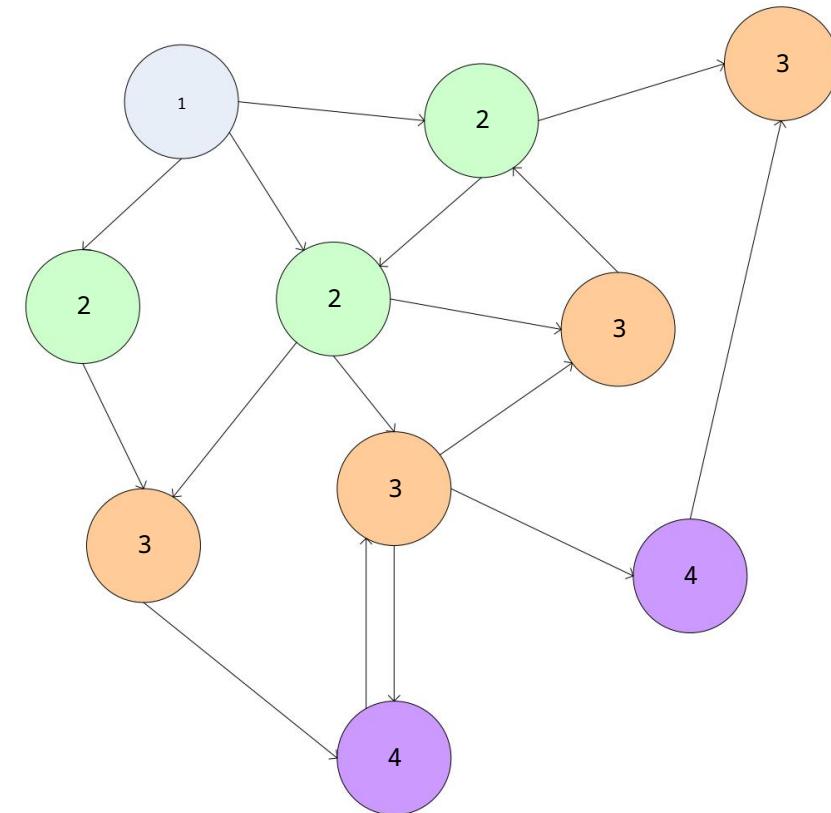
- Một số công việc nhỏ cộng lại thành một thuật toán đầy đủ
- Có thể tái sử dụng nhiều mã
 - Các lớp cổ phiếu tồn tại để tổng hợp, nhận dạng
- Công việc 3 và 4 thực sự có thể được thực hiện cùng lúc trong cùng một bộ giảm tốc, tiết kiệm chu kỳ ghi/đọc
- Rất dễ xử lý ở quy mô trung bình-lớn, nhưng phải cẩn thận để đảm bảo sử dụng bộ nhớ phẳng cho quy mô lớn nhất

Thuật toán tìm kiếm theo chiều rộng

- Thực hiện tính toán trên cấu trúc dữ liệu đồ thị đòi hỏi phải xử lý tại mỗi nút
- Mỗi nút chứa dữ liệu cụ thể của nút cũng như các liên kết (cạnh) đến các nút khác
- Tính toán phải đi qua đồ thị và thực hiện bước tính toán
- Làm thế nào để chúng ta duyệt một đồ thị trong MapReduce? Làm thế nào để chúng ta biểu diễn đồ thị cho việc này?

Tìm kiếm theo chiều rộng

- Tìm kiếm theo chiều rộng là một thuật toán lặp lại trên đồ thị
- Frontier tiến lên từ điểm xuất phát một cấp độ với mỗi lần vượt qua



Tìm kiếm theo chiều rộng & MapReduce

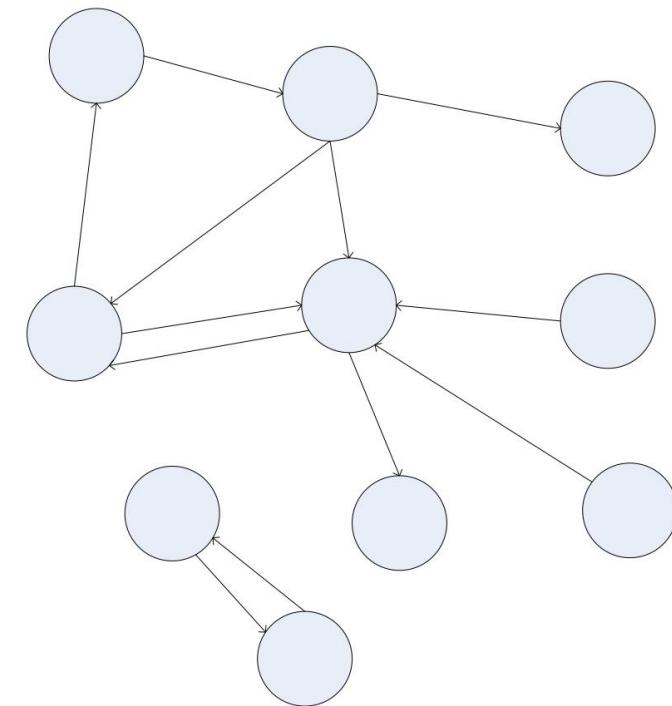
- Vấn đề
 - Điều này không “phù hợp” với MapReduce
- Giải pháp
 - Lặp lại các lần duyệt qua MapReduce - ánh xạ một số nút, kết quả bao gồm các nút bổ sung được đưa vào liên tiếp MapReduce vượt qua

Tìm kiếm theo chiều rộng & MapReduce

- Vấn đề
 - Gửi toàn bộ đồ thị đến một tác vụ bản đồ (hoặc hàng trăm/hàng nghìn tác vụ bản đồ) cần một lượng bộ nhớ khổng lồ
- Giải pháp
 - Cân nhắc cẩn thận cách chúng ta biểu diễn đồ thị

Biểu diễn đồ thị

- Biểu diễn trực tiếp nhất của đồ thị sử dụng các tham chiếu từ mỗi nút đến các nút lân cận của nó



Tham khảo trực tiếp

- Cấu trúc vốn có của đối tượng
- Lắp lại yêu cầu danh sách liên kết “xâu chuỗi” đồ thị
- Yêu cầu chế độ xem chung của bộ nhớ chia sẻ (đồng bộ hóa!)
- Không dễ dàng tuần tự hóa

```

lớp GraphNode
{
    Dữ liệu đối tượng;
    Vector<GraphNode>
        các cạnh ngoài;
    GraphNode lắp
        lại tiếp theo;
}
```

Ma trận kề

- Một biểu diễn đồ thị cổ điển khác. $M[i][j] = '1'$ ngụ ý một liên kết từ nút i đến nút j .
- Tự nhiên đóng gói lặp lại trên các nút

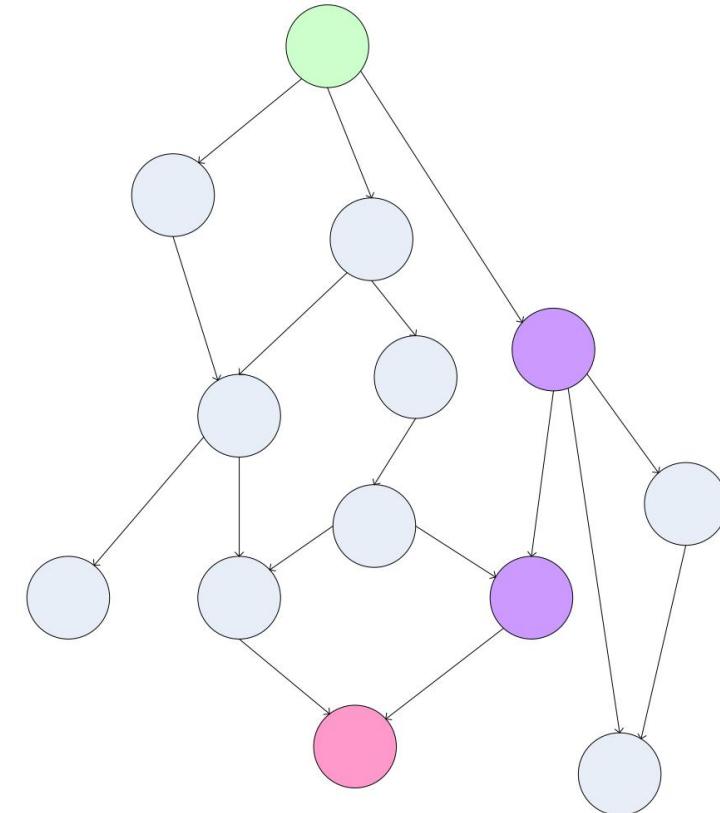
0	1	2	3	4
1	0	1	0	1
2	1	0	1	1
3	0	1	0	0
4	1	0	1	0

Ma trận kè: Biểu diễn thưa thớt

- Ma trận kè đối với hầu hết các đồ thị lớn (ví dụ: trang web) sẽ chứa đầy số không.
 - Mỗi hàng của đồ thị dài một cách vô lý
- Ma trận thưa thớt chỉ bao gồm các phần tử khác không
 - 1: (3, 1), (18, 1), (200, 1)
 - 2: (6, 1), (12, 1), (80, 1), (400, 1)
 - 3: (1, 1), (14, 1)
 - .
 - 1: 3, 18, 200
 - 2: 6, 12, 80, 400
 - 3: 1, 14
 - .

Tìm đường đi ngắn nhất

- Một ứng dụng tìm kiếm đồ thị phổ biến là tìm đường đi ngắn nhất từ một nút bắt đầu đến một hoặc nhiều nút đích
- Thường được thực hiện trên một máy duy nhất với Thuật toán Dijkstra
- Chúng ta có thể sử dụng BFS để tìm đường đi ngắn nhất thông qua MapReduce không?



Đây được gọi là bài toán đường dẫn ngắn nhất nguồn đơn. (hay còn gọi là SSSP)

Tìm đường đi ngắn nhất: Trực giác

- Chúng ta có thể xác định giải pháp cho vấn đề này theo phương pháp quy nạp:
 - $\text{DistanceTo}(\text{startNode}) = 0$
 - Đối với tất cả các nút n có thể truy cập trực tiếp từ startNode ,
Khoảng cách tới $(n) = 1$
 - Đối với tất cả các nút n có thể tiếp cận được từ một tập hợp các nút S khác,
 - $\text{DistanceTo}(n) = 1 + \min(\text{DistanceTo}(m), m \in S)$

Từ trực giác đến thuật toán

- Một tác vụ bắn đòn nhận một nút n làm khóa và (D, trả tới) làm giá trị của nó
 - D là khoảng cách đến nút từ điểm bắt đầu
 - points-to là danh sách các nút có thể tiếp cận được từ n
 - trả tới, phát ra (p, D+1)
- Giảm nhiệm vụ thu thập các khoảng cách có thể đến một p nhất định và chọn cái tối thiểu

Cuộc thảo luận

- Nhiệm vụ MapReduce này có thể tiến xa hơn biên giới đã biết bằng một bước nhảy
- Để thực hiện toàn bộ BFS, một non-MapReduce thành phần sau đó đưa đầu ra của bước này trở lại tác vụ MapReduce cho một lần lặp khác
 - Vấn đề: Danh sách điểm cần đạt được đã đi đâu?
 - Giải pháp: Mapper cũng phát ra (n , điểm-đến)

Thổi phồng và chấm dứt

- Thuật toán này bắt đầu từ một nút
- Các lần lặp tiếp theo bao gồm nhiều nút hơn của đồ thị khi biên giới tiến triển
- Điều này có bao giờ chấm dứt không?
 - Vâng! Cuối cùng, các tuyến đường giữa các nút sẽ ngừng hoạt động đã được khám phá và không có khoảng cách nào tốt hơn được tìm thấy. Khi khoảng cách là như nhau, chúng ta dừng lại
 - Người lập bản đồ phải phát ra (n, D) để đảm bảo rằng “khoảng cách hiện tại” là được đưa vào bộ giảm tốc

Thêm trọng lượng

- Đường dẫn ngắn nhất có trọng số hữu ích hơn chi phí cách tiếp cận
- Thay đổi đơn giản: danh sách điểm đến trong tác vụ bản đồ bao gồm trọng số 'w' cho mỗi nút được trả đến
 - phát ra ($p, D+wp$) thay vì ($p, D+1$) cho mỗi nút p
 - Hoạt động cho đồ thị có trọng số dương

So sánh với Dijkstra

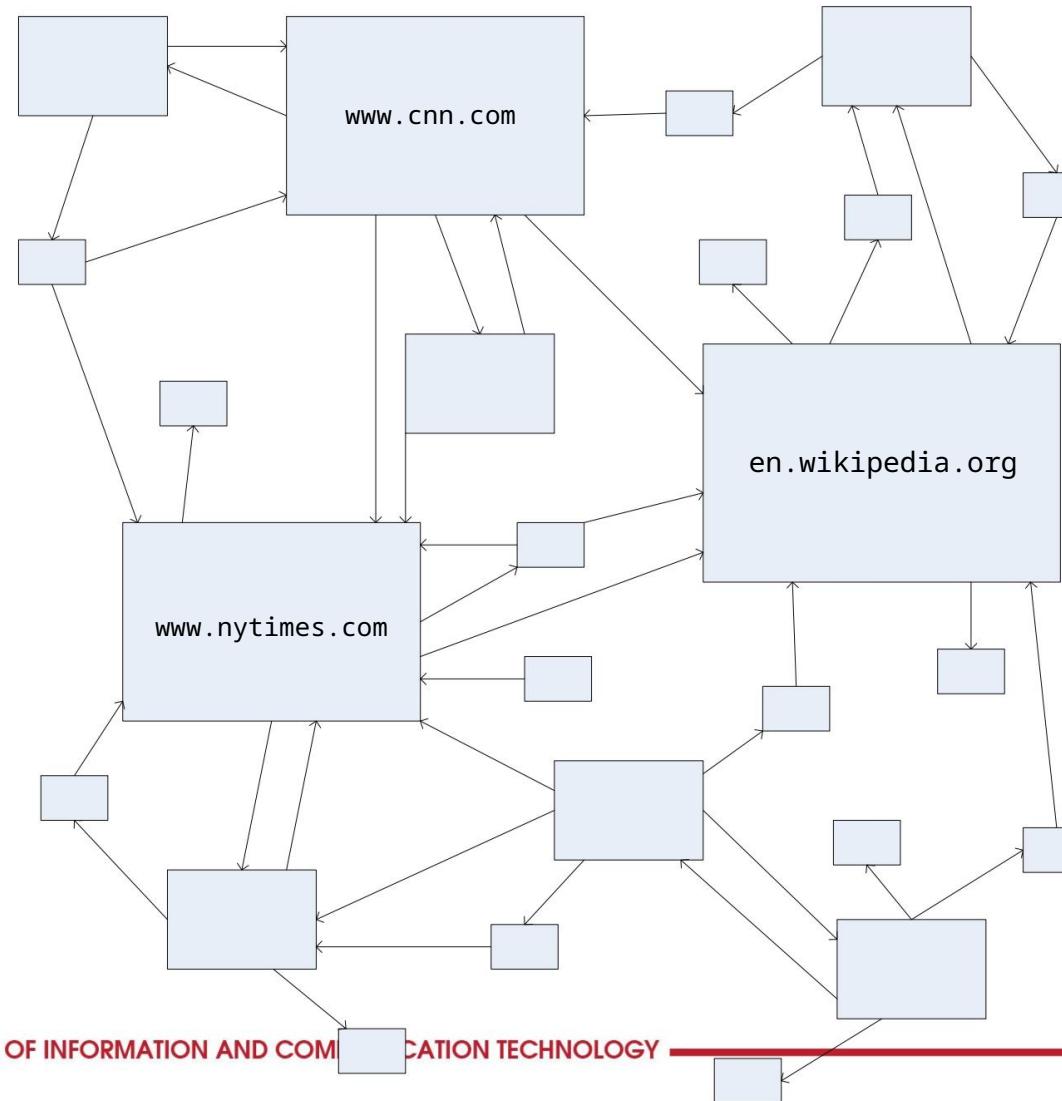
- Thuật toán Dijkstra hiệu quả hơn vì ở bất kỳ bước nào nó chỉ theo đuổi các cạnh từ đường có chi phí tối thiểu bên trong biên giới
- Phiên bản MapReduce khám phá tất cả các đường dẫn song song; nhìn chung không hiệu quả bằng, nhưng kiến trúc có khả năng mở rộng hơn
- Tương đương với Dijkstra về trọng số = 1 trường hợp

PageRank: Đi bộ ngẫu nhiên trên

Trang web

- Nếu người dùng bắt đầu ở một trang web ngẫu nhiên và lướt qua Khi nhấp vào liên kết và nhập ngẫu nhiên các URL mới, xác suất người đó sẽ đến được một trang nhất định là bao nhiêu?
- PageRank của một trang nắm bắt được khái niệm này
 - Các trang “phổ biến” hoặc “đáng giá” hơn sẽ có thứ hạng cao hơn

PageRank: Trực quan



PageRank: Công thức

- Cho trang A và các trang T1 đến Tn liên kết đến A, PageRank được định nghĩa như sau:
 - $$\text{PR}(A) = (1-d) + d \left(\frac{\text{PR}(T_1)}{C(T_1)} + \dots + \frac{\text{PR}(T_n)}{C(T_n)} \right)$$
- $C(P)$ là số lượng (bậc ra) của trang P
- d là hệ số giảm chấn ("URL ngẫu nhiên")

PageRank: Trực giác

- Tính toán là lặp đi lặp lại: PR_{i+1} dựa trên PR_i
- Mỗi trang phân phối PR_i của mình cho tất cả các trang mà nó liên kết tới. Người được liên kết cộng các mảnh xếp hạng đã trao của họ để tìm PR_{i+1} của họ
- d là một tham số có thể điều chỉnh (thường = 0,85) bao gồm "hệ số nhảy ngẫu nhiên"

$$\text{PR}(A) = (1-d) + d (\text{PR}(T_1)/C(T_1) + \dots + \text{PR}(T_n)/C(T_n))$$

PageRank: Triển khai đầu tiên

- Tạo hai bảng 'current' và 'next' chứa PageRank cho mỗi trang. Hạt giống 'hiện tại' với các giá trị PR ban đầu
- Lặp lại trên tất cả các trang trong biểu đồ, phân phối PR từ 'hiện tại' thành 'tiếp theo' của các liên kết
- hiện tại := tiếp theo; tiếp theo := bảng_mới();
- Quay lại bước lặp hoặc kết thúc nếu hội tụ

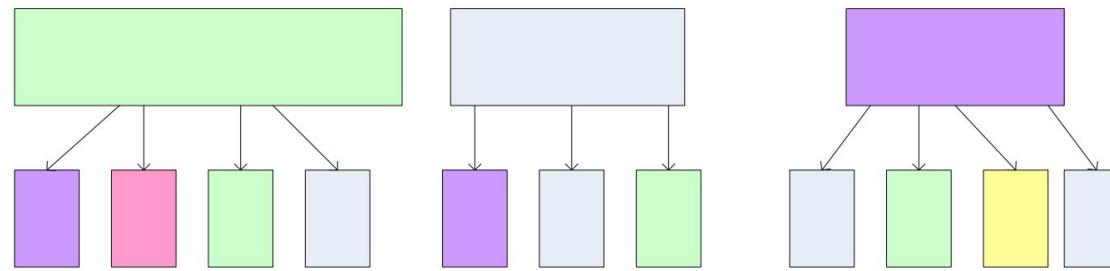
Phân phối thuật toán

- Những hiểu biết chính cho phép song song hóa:
 - Bảng 'tiếp theo' phụ thuộc vào 'hiện tại', nhưng không phụ thuộc vào bất kỳ hàng nào khác của 'tiếp theo'
 - Các hàng riêng lẻ của ma trận kè có thể được xử lý trong song song
 - Các hàng ma trận thưa thớt tương đối nhỏ

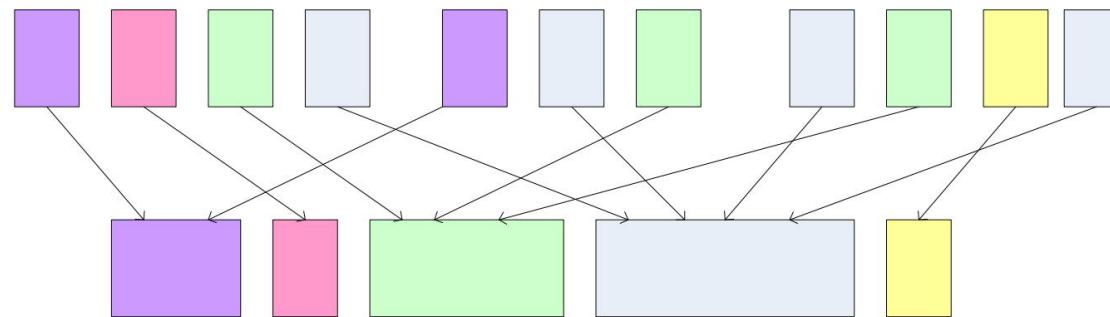
Phân phối thuật toán

- Hậu quả của sự hiểu biết sâu sắc:
 - Chúng ta có thể ánh xạ mỗi hàng của 'hiện tại' thành một danh sách PageRank "các đoạn" để gán cho người được liên kết
 - Những đoạn này có thể được giảm xuống thành một giá trị PageRank duy nhất cho một trang bằng cách cộng lại
 - Biểu diễn đồ thị có thể thậm chí còn nhỏ gọn hơn; vì mỗi phần tử chỉ đơn giản là 0 hoặc 1, chỉ truyền số cột khi nó là 1

Bước bắn đồ: chia thứ hạng trang thành các phần đều nhau để phân phối cho các mục tiêu liên kết



Bước giảm: thêm các đoạn vào PageRank tiếp theo



Lặp lại cho bước tiếp theo...

Giai đoạn 1: Phân tích cú pháp HTML

- Nhiệm vụ bản đồ lầy các cặp (URL, nội dung trang) và ánh xạ chúng tới (URL, (PRinit, danh sách các url))
 - PRinit là PageRank “hạt giống” cho URL • danh sách các url chứa tất cả các trang được trả tới bởi URL
- Giảm nhiệm vụ chỉ là chức năng nhận dạng

Giai đoạn 2: Phân phối PageRank

- Nhiệm vụ bản đồ chiếm (URL, (cur_rank, url_list))
 - Đối với mỗi u trong url_list, phát ra (u, cur_rank/|url_list|)
 - Phát ra (URL, url_list) để mang danh sách điểm đến qua lặp lại
- Giảm nhiệm vụ nhận được (URL, url_list) và nhiều (URL, val) giá trị
 - Tổng các giá trị và sửa chữa với d
 - Phát ra (URL, (xếp hạng mới, danh sách url))

$$\text{PR}(A) = (1-d) + d \left(\text{PR}(T_1)/C(T_1) + \dots + \text{PR}(T_n)/C(T_n) \right)$$

Kết thúc . . .

- Một thành phần tiếp theo xác định xem sự hội tụ đã đạt được (Số lần lặp cố định? So sánh các giá trị chính?)
- Nếu vậy, hãy viết ra danh sách PageRank - xong!
- Nếu không, đưa đầu ra của Pha 2 vào Pha khác 2 lần lặp lại

Nhận xét

- MapReduce chạy “công việc nặng nhọc” trong tính toán lặp lại
- Yếu tố chính trong song song hóa là độc lập Tính toán PageRank trong một bước nhất định
- Song song hóa đòi hỏi phải suy nghĩ về dữ liệu tối thiểu phân vùng để truyền (ví dụ, biểu diễn nhỏ gọn của các hàng đồ thị)
 - Ngay cả việc triển khai được trình bày ngày nay cũng không thực sự mở rộng ra toàn bộ Internet; nhưng nó hoạt động với các đồ thị có kích thước trung bình

Tài liệu tham khảo

- Dean, Jeffrey và Sanjay Ghemawat. "MapReduce: xử lý dữ liệu đơn giản trên các cụm lớn." *Communications of the ACM* 51.1 (2008): 107-113.
- Lin, Jimmy và Chris Dyer. "Xử lý văn bản chuyên sâu về dữ liệu với MapReduce." Bài giảng tổng hợp về công nghệ ngôn ngữ của con người 3.1 (2010): 1-177.
- Lee, Kyong-Ha, et al. "Xử lý dữ liệu song song với MapReduce: một cuộc khảo sát." *ACM SIGMOD Record* 40.4 (2012): 11-20.



25
YEARS ANNIVERSARY
SOICT

VIỆN CÔNG NGHỆ THÔNG TIN VÀ TRUYỀN THÔNG
SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

Cảm ơn sự
chú ý
của bạn!!!

