Parallel Programming with Hadoop/MapReduce

Slides from Tao Yang

1

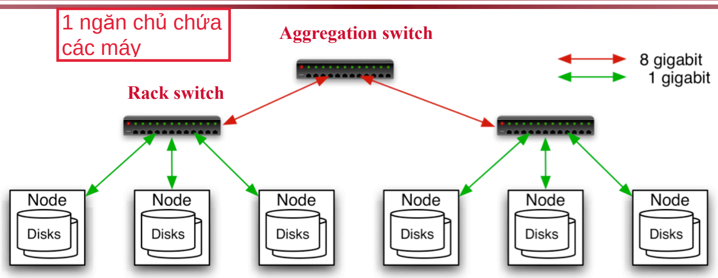
Overview

- **Related technologies**
  - –Hadoop/Google file system
- **MapReduce applications**

2

Typical Hadoop Cluster

1 ngăn chủ chứa các máy



- 40 nodes/rack, 1000-4000 nodes in cluster
- 1 Gbps bandwidth in rack, 8 Gbps out of rack
- Node specs :
  8-16 cores, 32 GB RAM, 8×1.5 TB disks

5

kp ngôn ngữ lập trình, như kiểu interface

MapReduce Programming Model

• Lấy cảm hứng từ bản đồ và giảm các hoạt động thường được sử dụng trong các ngôn ngữ lập trình chức năng như Lisp.
• Có nhiều nhiệm vụ bản đồ và giảm bớt nhiệm vụ
• Người dùng triển khai giao diện của hai phương thức chính:

- Inspired from map and reduce operations commonly used in functional programming languages like Lisp.
- Have multiple map tasks and reduce tasks
- Users implement interface of two primary methods:
  - Map: (key1, val1) → (key2, val2)
  - Reduce: (key2, [val2]) → [val3]

Node

có nhiều

Jobs

có nhiều

(key,value)

Task

Tasks = process

7

Example: Map Processing in Hadoop

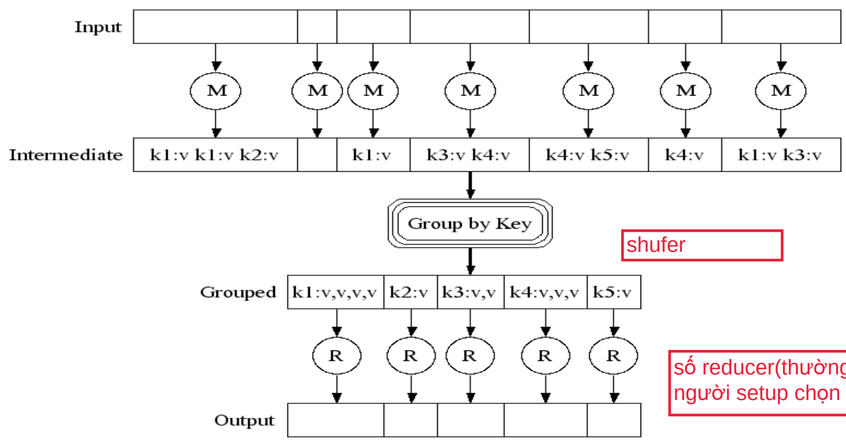
- **Given a file**
  - A file may be divided into multiple parts (splits).

chunks -> số mapper

- **Each record (line) is processed by a Map function,**
  - written by the user,
  - takes an input key/value pair
  - produces a set of intermediate key/value pairs.
  - e.g. (doc—id, doc-content)
- **Draw an analogy to SQL *group-by* clause**

8

Put Map and Reduce Tasks Together

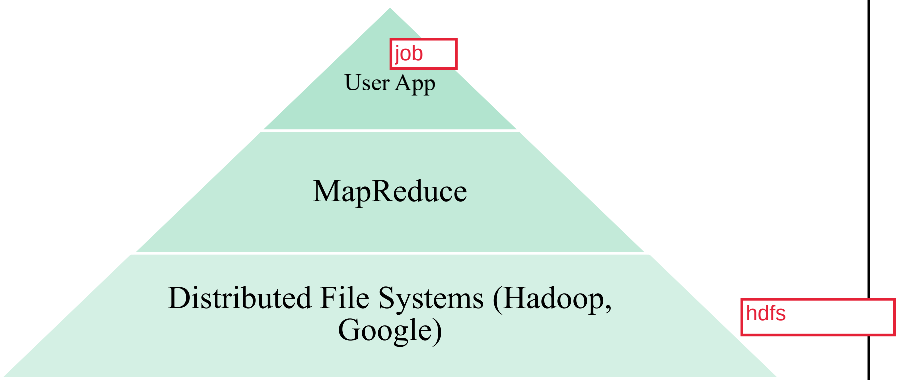


shufer

số reducer(thường ko tính toán nhiều)
người setup chọn ~ số máy có

12

Systems Support for MapReduce



20

Distributed Filesystems

- **The interface is the same as a single-machine file system**
  – create(), open(), read(), write(), close()
- **Distribute file data to a number of machines (storage units).**
  – Support replication
- **Support concurrent data access**
  – Fetch content from remote servers. Local caching
- **Different implementations sit in different places on complexity/feature scale**
  – Google file system and Hadoop HDFS
    » Highly scalable for large data-intensive applications.
    » Provides redundant storage of massive amounts of data on cheap and unreliable computers

21

bỏ

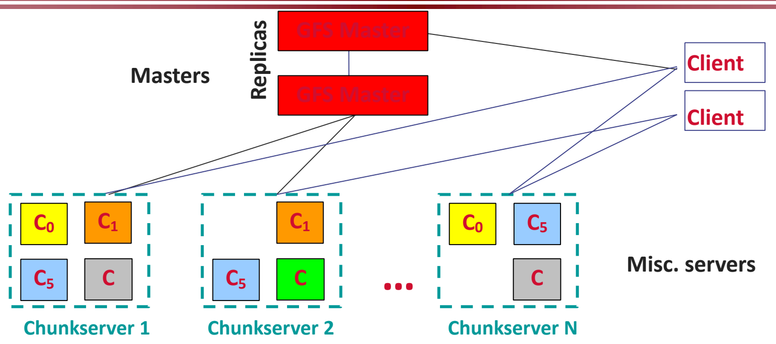
Assumptions of GFS/Hadoop DFS

google file systems

- **High component failure rates**
  - Inexpensive commodity components fail all the time
- **"Modest" number of HUGE files**
  - Just a few million
  - Each is 100MB or larger; multi-GB files typical
- **Files are write-once, mostly appended to**
  - Perhaps concurrently
- **Large streaming reads**
- **High sustained throughput favored over low latency**

22

GFS Design



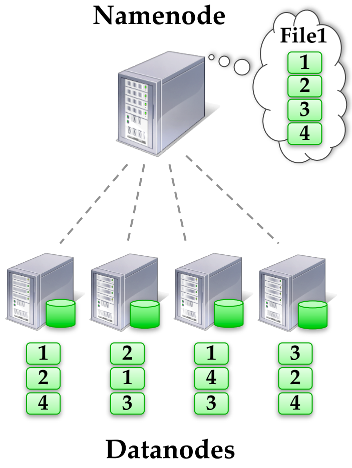
- **Files are broken into chunks (typically 64 MB) and serve in chunk servers**
- **Master manages metadata, but clients may cache meta data obtained.**
  - **Data transfers happen directly between clients/chunk-servers**
- **Reliability through replication**

  Each chunk replicated across 3+ *chunk-servers*

23

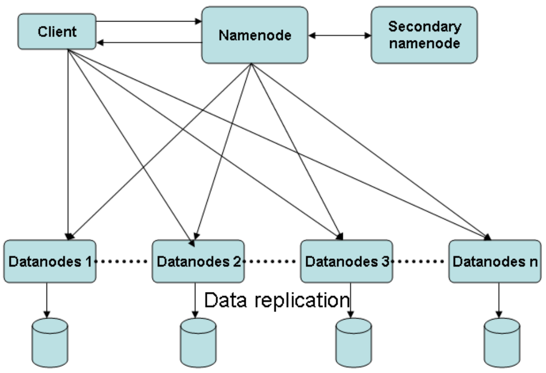
Hadoop Distributed File System

- Files split into 128MB blocks
- Blocks replicated across several datanodes (often 3)
- Namenode stores metadata (file names, locations, etc)
- Optimized for large files, sequential reads
- Files are append-only



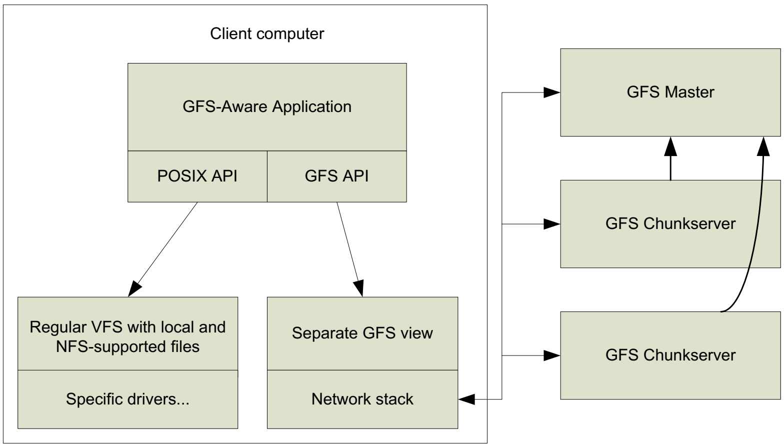
24

Hadoop DFS



25

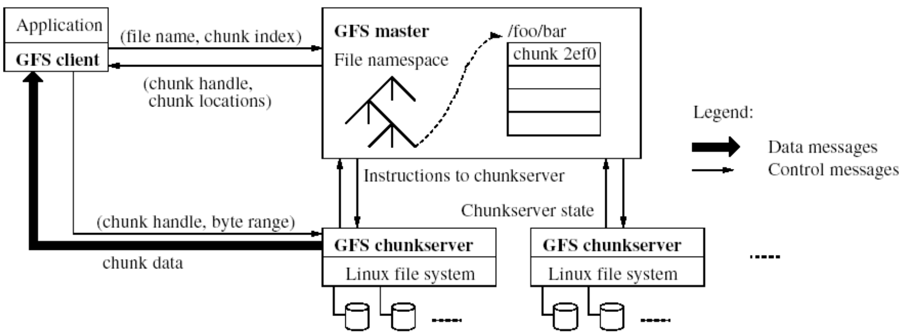
GFS Client Block Diagram



- **Provide both POSIX standard file interface, and costumed API**
- **Can cache meta data for direct client-chunk server access**

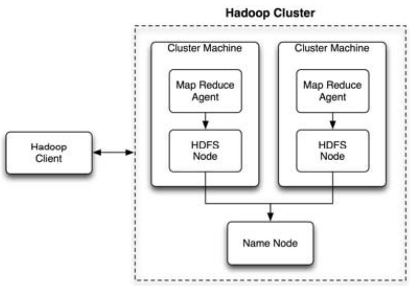
26

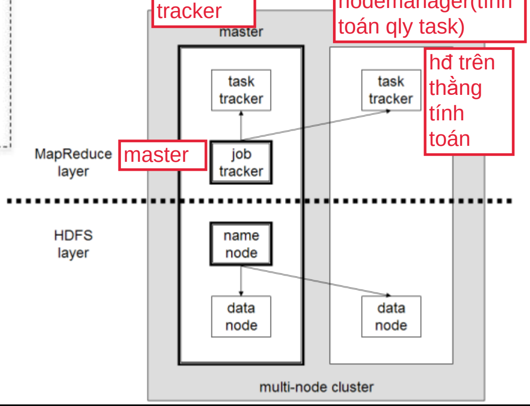
Read/write access flow in GFS



27

Hadoop DFS with MapReduce





hadoop 1.0 (qly TN ko tốt) job, task tracker

hadoop 2.0: sd YARN -> resource manage(qly jobtracker) và nodemanager(tính toán qly task)

hđ trên thằng tính toán

master

28

MapReduce: Execution overview

Master Server distributes M map tasks to machines and monitors their progress.

tự chia dl thành M maptask cho các máy trong mạng

Map task reads the allocated data, saves the map results in local buffer.

Shuffle phase assigns reducers to these buffers, which are remotely read and processed by reducers.

Reducers output the result on stable storage.

29

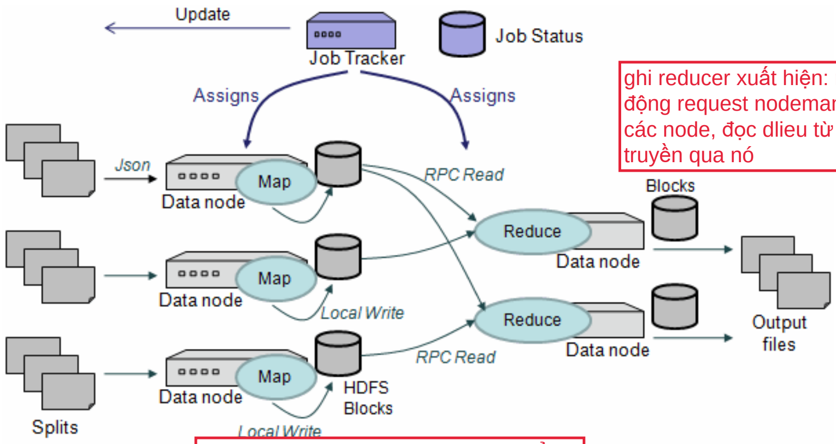
shufer: ghép cùng key về 1 chỗ

Execute MapReduce on a cluster of machines with Hadoop DFS

hadoop1.0, gửi nhận thông qua RPC(remote proxy call)

hadoop2.0:gửi/ nhận dlieu: HTTP

ghi reducer xuất hiện: nó chủ động request nodemanager ở các node, đọc dlieu từ ext4 truyền qua nó



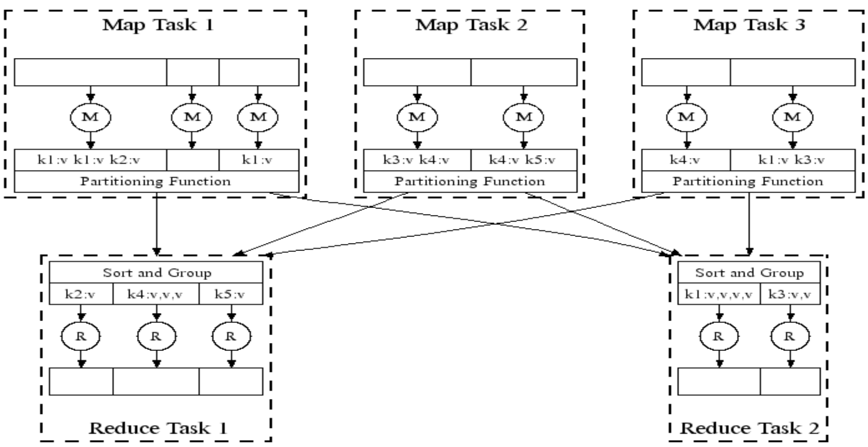
output reduce: ghi vào HDFS cho job khác

output mapper: spinout vào buffer để ghi vào disk cứng của máy (ext4:ubuntu)(ko ghi vào HDFS vì sẽ tạo ra bản sao) => hạn chế trong TH máy cứng bị chết

30

TH vẫn còn 2 task chờ Mapper kai đang xử lý tasks khác: Setup time chờ, nếu quá split qua máy khác để xử lý, ko thì chờ Mapper trên cùng máy

MapReduce in Parallel: Example



partition: trc khi vào shufer, gộp các mess cùng key lại r ms gửi

31

MapReduce: Execution Details

- **Input reader**
  – Divide input into splits, assign each split to a Map task
- **Map task**
  – Apply the Map function to each record in the split
  – Each Map function returns a list of (key, value) pairs
- **Shuffle/Partition and Sort**
  – Shuffle distributes sorting & aggregation to many reducers
  – All records for key *k* are directed to the same reduce processor
  – Sort groups the same keys together, and prepares for aggregation
- **Reduce task**
  – Apply the Reduce function to each key
  – The result of the Reduce function is a list of (key, value) pairs

Đầu đọc đầu vào
- Chia đầu vào thành các phần, gán mỗi phần cho một Map task

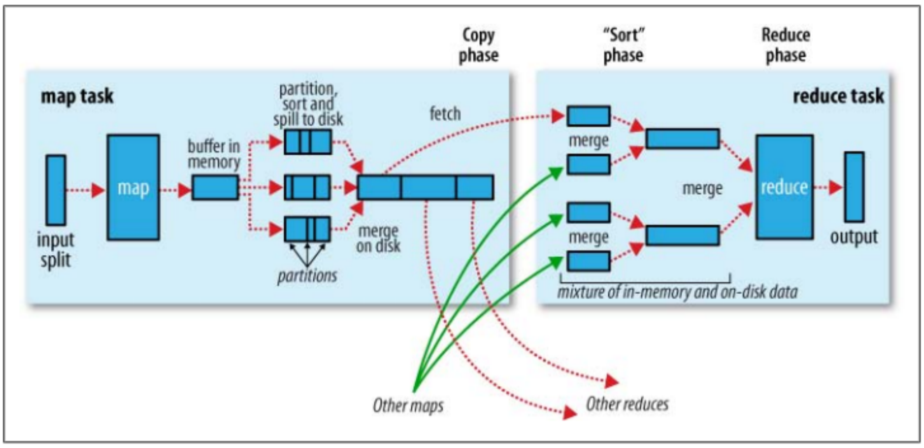
Map task
- Áp dụng hàm Map cho từng bản ghi trong phần tách
- Mỗi hàm Map trả về một danh sách các cặp (khóa, giá trị)

Trộn / phân vùng và sắp xếp
- Shuffle phân phối sắp xếp và tổng hợp thành nhiều bộ giảm
- Tất cả các bản ghi cho khóa k được chuyển hướng đến cùng một bộ xử lý rút gọn
- Sắp xếp các nhóm có cùng khóa với nhau và chuẩn bị cho việc tổng hợp

Reduce task
- Áp dụng chức năng Giảm cho từng phím
- Kết quả của hàm Reduce là danh sách các cặp (key, value)

32

MapReduce with data shuffling& sorting



Tom White, *Hadoop: The Definitive Guide*

33

Lập lịch chương trình trên toàn bộ cụm
máy móc, Tối ưu hóa vị trí và Cân bằng tải

MapReduce: Runtime Environment &Hadoop

lq đến tính chịu lỗi cao

phân vùng

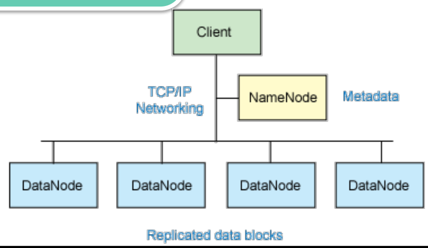
Partitioning the input data.

Scheduling program across cluster of machines, Locality Optimization and Load balancing

MapReduce Runtime Environment

Dealing with machine failure

Managing Inter-Machine communication

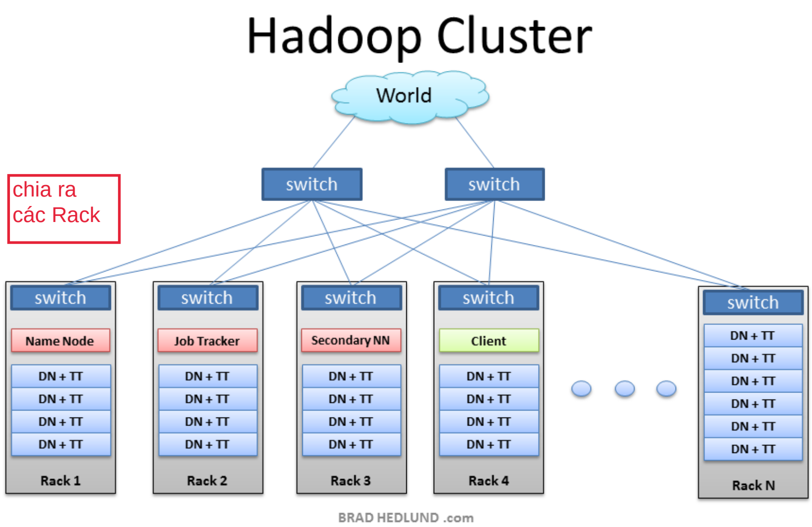


Các tính chất:
- Locality
- Load balancer: Chia để mapper cho các máy, tối đa bao nhiêu
- Scheduling: (khởi động Reducer, khi mà output của mapper xuất hiện và reducer có nơi để chạy)

+fault tolerance: tính chịu lỗi cao(trên 1 máy, 1 jobs, 1 task)
+inter-machine comunity: HTTP, RPC

34

Hadoop Cluster with MapReduce



chia ra các Rack

35

đảm bảo tính chịu lỗi cao

MapReduce: Fault Tolerance

Xử lý thông qua thực hiện lại các tác vụ.
- Hoàn thành nhiệm vụ được cam kết thông qua tổng thể

- **Handled via re-execution of tasks.**
  - Task completion committed through master

TH chạy xong map, chết khi đã ghi dl ra disk, reduce chưa biết -> khởi động lại map

Mapper lưu kết quả đầu ra vào đĩa cục bộ trước khi phân phát tới reducer
- Cho phép khôi phục nếu reduce gặp sự cố
- Cho phép chạy nhiều reducer hơn số nút

- **Mappers save outputs to local disk before serving to reducers**
  - Allows recovery if a reducer crashes
  - Allows running more reducers than # of nodes
- **If a task crashes:**
  - Retry on another node
    - » OK for a map because it had no dependencies
    - » OK for reduce because map outputs are on disk
  - If the same task repeatedly fails, fail the job or ignore that input block
  - : For the fault tolerance to work, *user tasks must be deterministic and side-effect-free*

khi map task chết:
khởi động lại trên máy1(1,2 lần nếu vẫn chết) thì dịch chuyển sang máy 2

khi reducer chết:
-khi quét xong dl
- khi chưa quét xong
đều restart lại (trong qtr đang khi mà chết, dl buffer vẫn còn, khi nào ghi ra HDFS done ms xóa buffer)

- job chết(khi quá nhiều task chết >70%)

2. If a node crashes:
- Relaunch its current tasks on other nodes
- Relaunch any maps the node previously ran
  - » Necessary because their output files were lost along with the crashed node

khi 1 node chết -> thực hiện rebalance -> tiến hành dịch chuyển

36

1000 máy, hadoop nhanh hơn spark

MapReduce: Locality Optimization

đạt được khi có nhiều bản sao

- Leverage the distributed file system to schedule a map task on a machine that contains a replica of the corresponding input data.
- Thousands of machines read input at local disk speed
- Without this, rack switches limit read rate

Tận dụng hệ thống tệp phân tán để lên lịch map task trên máy có chứa bản sao của dữ liệu đầu vào tương ứng.

• Hàng nghìn máy đọc đầu vào ở tốc độ đĩa cục bộ

• Nếu không có điều này, công tắc giá đỡ giới hạn tốc độ đọc

37

MapReduce: Redundant Execution

- Slow workers are source of bottleneck, may delay completion time.
- Near end of phase, spawn backup tasks, one to finish first wins.
- Effectively utilizes computing power, reducing job completion time by a factor.

38

MapReduce: Skipping Bad Records

- Map/Reduce functions sometimes fail for particular inputs.
- Fixing the Bug might not be possible : Third Party Libraries.
- On Error
  - Worker sends signal to Master
  - If multiple error on same record, skip record

39

MapReduce: Miscellaneous Refinements

- Combiner function at a map task
- Sorting Guarantees within each reduce partition.
- Local execution for debugging/testing
- User-defined counters

40

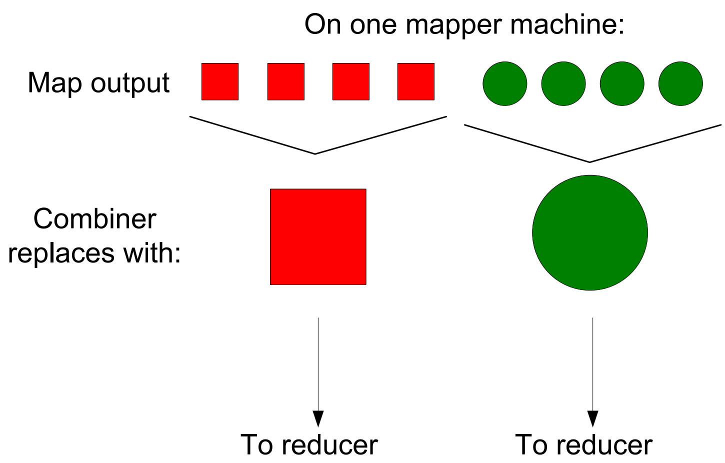
giữa mapping và shufer
function gộp các key trước khi chuyển cho shufer

Combining Phase

- Run on map machines after map phase
- “Mini-reduce,” only on local map output
- Used to save bandwidth before sending data to full reduce tasks
- Reduce tasks can be combiner if commutative & associative

41

Combiner, graphically

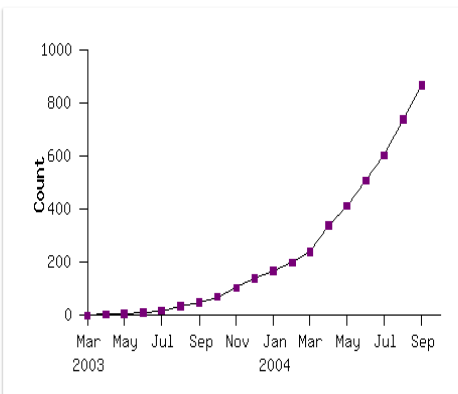


42

Examples of MapReduce Usage in Web Applications

- Distributed Grep.
- Count of URL Access Frequency.
- Clustering (K-means)
- Graph Algorithms.
- Indexing Systems

MapReduce Programs In Google Source Tree



43

Hadoop and Tools

- **Various Linux Hadoop clusters around**
  - Cluster +Hadoop
    - » http://hadoop.apache.org
  - Amazon EC2
- **Winows and other platforms**
  - The NetBeans plugin simulates Hadoop
  - The workflow view works on Windows
- **Hadoop-based tools**
  - For Developing in Java, NetBeans plugin
- **Pig Latin,** a SQL-like high level data processing script language
- **Hive,** Data warehouse, SQL
- **Mahout,** Machine Learning algorithms on Hadoop
- **HBase,** Distributed data store as a large table

44

More MapReduce Applications

- Map Only processing
- Filtering and accumulation
- Database join
- Reversing graph edges
- Producing inverted index for web search
- PageRank graph processing

45

MapReduce Use Case 1: Map Only

Data distributive tasks – Map Only

- **E.g. classify individual documents**
- **Map does everything**
  - Input: (docno, doc_content), …
  - Output: (docno, [class, class, …]), …
- **No reduce tasks**

46

hadoop đơn giản tính offline (ko realtime đc như spark streaming)

MapReduce Use Case 2: Filtering and Accumulation

tìm min, max, sum, tìm gtr,...

Filtering & Accumulation – Map and Reduce

- **E.g. Counting total enrollments of two given student classes** VD: bn người đki môn
- **Map** selects records and outputs initial counts
  - In: (Jamie, 11741), (Tom, 11493), …
  - Out: (11741, 1), (11493, 1), …
- **Shuffle/Partition** by class_id
- **Sort**
  - In: (11741, 1), (11493, 1), (11741, 1), …
  - Out: (11493, 1), …, (11741, 1), (11741, 1), …
- **Reduce accumulates counts**
  - In: (11493, [1, 1, …]), (11741, [1, 1, …])
  - Sum and Output: (11493, 16), (11741, 35)

47

join trong
nosql: tốn TN

MapReduce Use Case 3: Database Join

- **A JOIN is a means for combining fields from two tables by using values common to each.**
- **Example :For each employee, find the department he works in**

Employee Table	
LastName	**DepartmentID**
Rafferty	31
Jones	33
Steinberg	33
Robinson	34
Smith	34

JOIN

Pred:

EMPLOYEE.DepID=
DEPARTMENT.DepID

Department Table	
DepartmentID	**DepartmentName**
31	Sales
33	Engineering
34	Clerical
35	Marketing

JOIN RESULT	
LastName	**DepartmentName**
Rafferty	Sales
Jones	Engineering
Steinberg	Engineering
…	…

Map:
- In: (name,departmentID) (departmentID,departmantName)
- Out: (departmentID,name) (departmentID,departmantName)
Reduce:
- In: (departmentID,[name,departmantName]),...
-Out: (name,departmantName)

48

MapReduce Use Case 3 – Database Join

Problem: Massive lookups
- Given two large lists: (URL, ID) and (URL, doc_content) pairs
- Produce (URL, ID, doc_content) or (ID, doc_content)

Solution:
- **Input stream**: both (URL, ID) and (URL, doc_content) lists
  - (http://del.icio.us/post, 0), (http://digg.com/submit, 1), …
  - (http://del.icio.us/post, <html0>), (http://digg.com/submit, <html1>), …
- **Map** simply passes input along,
- **Shuffle and Sort on URL** (group ID & doc_content for the same URL together)
  - Out: (http://del.icio.us/post, 0), (http://del.icio.us/post, <html0>), (http://digg.com/submit, <html1>), (http://digg.com/submit, 1), …
- **Reduce** outputs result stream of (ID, doc_content) pairs
  - In: (http://del.icio.us/post, [0, html0]), (http://digg.com/submit, [html1, 1]), …
  - Out: (0, <html0>), (1, <html1>), …

49

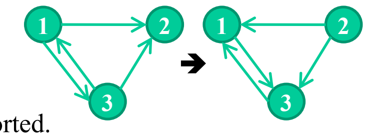
đảo ngược cạnh đồ thị có hướng và đầu ra theo thứ tự node

MapReduce Use Case 4: Reverse graph edge directions & output in node order

(đỉnh, hàng xóm) -> (đỉnh, hàng xóm mới)

- **Input example: adjacency list of graph (3 nodes and 4 edges)**

  (3, [1, 2]) (1, [3])
  (1, [2, 3]) ➔ (2, [1, 3])
  (3, [1])



- node_ids in the output **values** are also sorted. But Hadoop only sorts on keys!
- **MapReduce format**

Map:
- In: (verID, [v1,v2,..])
- Out: (v1, verID), (v2, verID),...

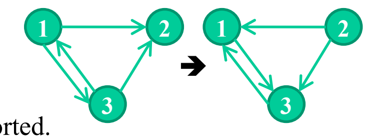
Reduce:
- In: (v1, verID), (v2, verID),...
- Out: (v1, [verID1,verID2]),...

50

MapReduce Use Case 4: Reverse graph edge directions & output in node order

- **Input example: adjacency list of graph (3 nodes and 4 edges)**

  (3, [1, 2]) (1, [3])
  (1, [2, 3]) ➔ (2, [1, 3])
  (3, [1])



- node_ids in the output **values** are also sorted. But Hadoop only sorts on keys!
- **MapReduce format**
  - Input: (3, [1, 2]), (1, [2, 3]).
  - Intermediate: (1, [3]), (2, [3]), (2, [1]), (3, [1]). (reverse edge direction)
  - Out: (1,[3]) (2, [1, 3]) (3, [[1]).

51

Sơ bộ lập chỉ mục ngược

MapReduce Use Case 5: Inverted Indexing Preliminaries

Xây dựng danh sách đảo ngược để tìm kiếm tài liệu

Construction of inverted lists for document search

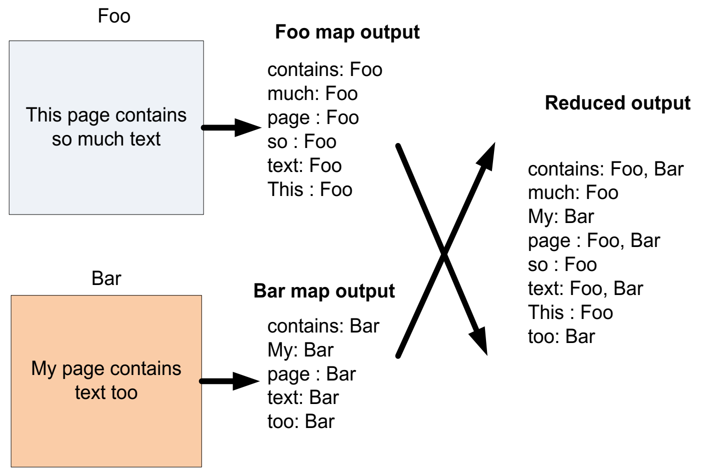
- Input: documents: (docid, [term, term..]), (docid, [term, ..]), ..
- Output: (term, [docid, docid, …])
  - E.g., (apple, [1, 23, 49, 127, …])

A document id is an internal document id, e.g., a unique integer

- Not an external document id such as a url

52

Inverted Index: Data flow

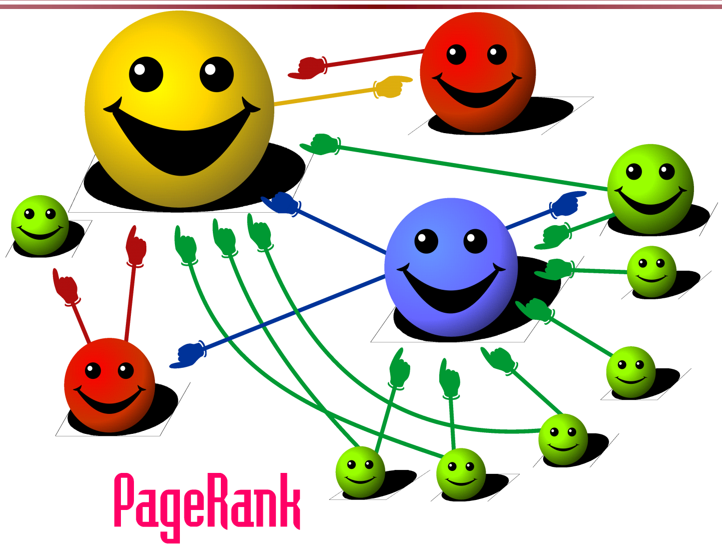


54

Map:
- In: (docID, [tem,...])
- Out: (tem, docID),...

Reduce:
-In: (tem, docID),...
-Out: (tem, [docID])

MapReduce Use Case 6: PageRank



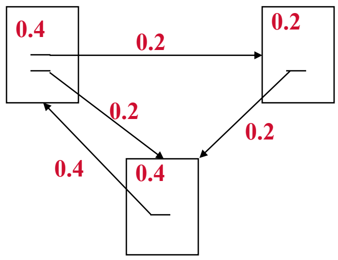
60

PageRank

- Model page reputation on the web

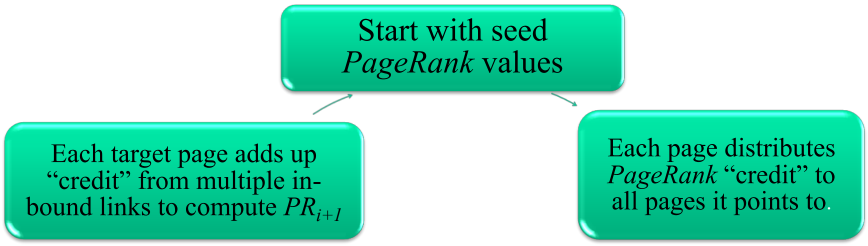
$$PR(x) = (1-d) + d\sum_{i=1}^{n}\frac{PR(t_i)}{C(t_i)}$$

- i=1,n lists all parents of page x.
- PR(x) is the page rank of each page.
- C(t) is the out-degree of t.
- d is a damping factor .



61

Computing PageRank Iteratively

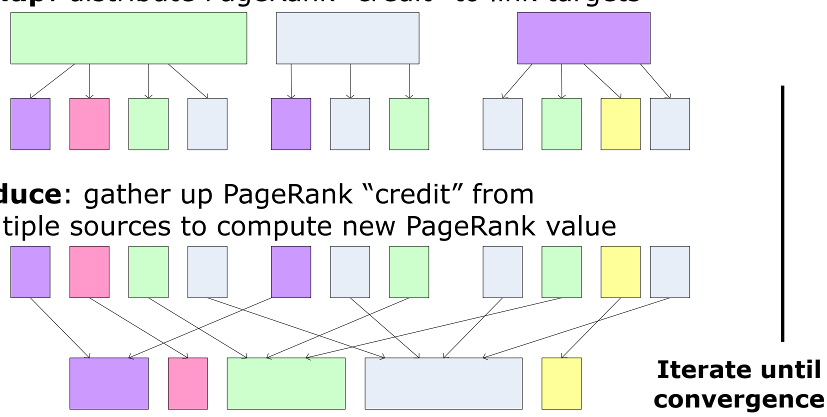


- Effects at each iteration is local. i+1th iteration depends only on ith iteration
- At iteration i, PageRank for individual nodes can be computed independently

PageRank using MapReduce

Map: distribute PageRank "credit" to link targets

Reduce: gather up PageRank "credit" from multiple sources to compute new PageRank value



Source of Image: Lin 2008

PageRank Calculation: Preliminaries

One PageRank iteration:

- Input:
  - $(id_1, [score_1^{(t)}, out_{11}, out_{12}, ..]), (id_2, [score_2^{(t)}, out_{21}, out_{22}, ..])$ ..
- Output:
  - $(id_1, [score_1^{(t+1)}, out_{11}, out_{12}, ..]), (id_2, [score_2^{(t+1)}, out_{21}, out_{22}, ..])$ ..

MapReduce elements

- Score distribution and accumulation
- Database join

64

PageRank: Score Distribution and Accumulation

- **Map**
  - In: $(id_1, [score_1^{(t)}, out_{11}, out_{12}, ..]), (id_2, [score_2^{(t)}, out_{21}, out_{22}, ..])$ ..
  - Out: $(out_{11}, score_1^{(t)}/n_1), (out_{12}, score_1^{(t)}/n_1)$ .., $(out_{21}, score_2^{(t)}/n_2)$, ..
- **Shuffle & Sort by node_id**
  - In: $(id_2, score_1), (id_1, score_2), (id_1, score_1)$, ..
  - Out: $(id_1, score_1), (id_1, score_2)$, .., $(id_2, score_1)$, ..
- **Reduce**
  - In: $(id_1, [score_1, score_2, ..]), (id_2, [score_1, ..])$, ..
  - Out: $(id_1, score_1^{(t+1)}), (id_2, score_2^{(t+1)})$, ..

65

PageRank: Database Join to associate outlinks with score

- **Map**
  - In & Out: (id_1, $score_1^{(t+1)}$), (id_2, $score_2^{(t+1)}$), .., (id_1, [out_{11}, out_{12}, ..]), (id_2, [out_{21}, out_{22}, ..]) ..
- **Shuffle & Sort by node_id**
  - Out: (id_1, $score_1^{(t+1)}$), (id_1, [out_{11}, out_{12}, ..]), (id_2, [out_{21}, out_{22}, ..]), (id_2, $score_2^{(t+1)}$), ..
- **Reduce**
  - In: (id_1, [$score_1^{(t+1)}$, out_{11}, out_{12}, ..]), (id_2, [out_{21}, out_{22}, .., $score_2^{(t+1)}$]), ..
  - Out: (id_1, [$score_1^{(t+1)}$, out_{11}, out_{12}, ..]), (id_2, [$score_2^{(t+1)}$, out_{21}, out_{22}, ..]) ..

66

Conclusions

- **MapReduce advantages**
- **Application cases**
  - Map only: for totally distributive computation
  - Map+Reduce: for filtering & aggregation
  - Database join: for massive dictionary lookups
  - Secondary sort: for sorting on values
  - Inverted indexing: combiner, complex keys
  - PageRank: side effect files

67

For More Information

- J. Dean and S. Ghemawat. “MapReduce: Simplified Data Processing on Large Clusters.” *Proceedings of the 6th Symposium on Operating System Design and Implementation (OSDI 2004)*, pages 137-150. 2004.
- S. Ghemawat, H. Gobioff, and S.-T. Leung. “The Google File System.” *OSDI 200?*
- http://hadoop.apache.org/common/docs/current/mapred_tutorial.html. “Map/Reduce Tutorial”. Fetched January 21, 2010.
- Tom White. *Hadoop: The Definitive Guide*. O'Reilly Media. June 5, 2009
- http://developer.yahoo.com/hadoop/tutorial/module4.html
- J. Lin and C. Dyer. *Data-Intensive Text Processing with MapReduce*. Book Draft. February 7, 2010.