Hadoopで行う大規模データ処理

kzk <kzk@preferred.jp>

Hadoopとは？

- Googleの基盤ソフトウェアのクローン
  - Google File System
  - MapReduce
- Yahoo Research の Doug Cutting氏が開発
  - 元々はNutch Crawlerのサブプロジェクト
  - Dougの子供の持っているぬいぐるみの名前
- Javaで記述
- Amazon S3との親和性〇

Google関連 参考論文 & スライド

- The Google File System
  - Sanjay Ghemawat, Howard Gobioff, and Shu-Tak Leong, SOSP 2003
- MapReduce: Simplified Data Processing on Large Clusters
  - Jeffrey Dean and Sanjay Ghemawat, SOSP 2004
- Parallel Architectures and Compilation Techniques (PACT) 2006, KeyNote
  - http://www.cs.virginia.edu/~pact2006/program/mapreduce-pact06-keynote.pdf

Hadoop参考文献

- Hadoop公式サイト
  - http://hadoop.apache.org/core/
  - Wiki: http://wiki.apache.org/hadoop/
    - インストール方法・チュートリアル・プレゼン資料など
- Hadoop解析資料
  - http://preferred.jp/pub/hadoop.html
- Hadoop, hBaseで構築する大規模データ処理システム on Codezine
  - http://codezine.jp/a/article/aid/2448.aspx

Hadoop解析資料 with NTT-Rさん

- OSS分散システムHadoopの検証
  - プロダクションに投入できるレベルか？
  - Googleの実装との比較
    - 機能単位で検証
  - ソースコードの解析
    - ソフトウェアの大体の構造
    - 重要な部分を詳しく検証
  - ベンチマーク
    - スケーラビリティを確認
  - 結果はオープンに公開
    - http://preferred.jp/pubs/hadoop.html

調査結果概要

- Googleインフラの機能は大体実装済み
  - アトミックな追記, エラーレコードスキップもv0.19.0で実装される
    - svnにはcommit済みで、RCがMLに流れている
    - しかし大幅に変更が入ったため、(たぶん)しばらくは不安定。あと1年程度で成熟期か?
- スケーラビリティ
  - 12台程度まではスケール
- 安定性にはまだ疑問
  - レプリケーション数3の時にジョブが失敗する, etc.

Hadoop周りのニュース

- Scaling Hadoop to 4000 nodes
  - http://developer.yahoo.net/blogs/hadoop/2008/09/scaling_hadoop_to_4000_nodes_a.html
  - いくつかのバグを潰すことによって500 nodesの4～7倍の性能を発揮
    - pingのタイミングが引き起こすバグなど

Hadoop使用事例

国外の使用事例

- Yahoo
  - ～2000ノード
  - 検索、広告、ログ処理、データ解析、etc
  - SIGIRなどでもY!Rの論文にはHadoopが出てくる
- Amazon, Facebook
  - ～400ノード
  - ログ処理、データ解析
- その他
  - 行動ターゲティング、検索インデクシング等

国内の採用事例

- はてな
  - ログ解析
  - はてなブックマーク2のバックエンドで使用
    - 全文検索まわり
- 楽天
  - 大規模レコメンデーションエンジン
  - http://www.atmarkit.co.jp/news/200812/01/rakuten.html
- メールで何件か相談
  - 検索系(Lucene)、ログ処理系が多い
  - ～100台
    - Cellクラスター, EC2 (blogeye)等も

MapReduce

Motivation

問題

- インターネットの爆発的普及により、非常に大規模なデータが蓄積されている
  - 例えばWebページを考えて見る。
    - 200億ページ * 20KB = 400 TB
  - Disk読み込み性能は50MB/sec (SATA)
    - 1台では読み込むだけでも約100日
    - 保存するだけでも1000個程度のDiskが必要
- このデータを効率的に処理したい

解決方法

- お金
- とにかく大量のマシンを用意
  - 1000台マシンがあれば1台で400G処理すればok
  - 読み込むのに8000秒程度で済む

お金だけでは解決しない

- プログラミングが非常に困難になる
  - プロセス起動
  - プロセス監視
  - プロセス間通信
  - デバッグ
  - 最適化
  - 故障時への対応
- しかも、新しいプログラムを作る度にこれらの問題をいちいち実装する必要がある

既存の分散/並列プログラミング環境

- MPI (Message Passing Interface)
  - 並列プログラミングのためのライブラリ
    - スパコンの世界では主流
  - プログラマは各プロセスの挙動を記述
    - 通信プリミティブ(Send, Recv, All-to-All)が提供されており、それを用いてデータ通信を実現
  - 利点
    - 通信パターンなどをプログラマがコントロールでき、問題に対して最適なプログラムを記述する事ができる

MPIの問題点

- 問題点
  - 耐障害性への考慮が少ない
    - アプリケーションが独自にチェックポイント機能を実装
    - 1万台以上の環境で計算するには耐えられない
      - 1台が1000日程度で壊れるとすると、1日で10台程度壊れる
      - 壊れる度にチェックポイントから戻すとかやってらんない
      - RAID組んでもそのうち壊れるので一緒
  - 通信パターンなどを記述する作業が多くなり、実際のアルゴリズムを記述するのにたどり着くまで時間がかかる

Googleでの使用率

Usage Statistics Over Time

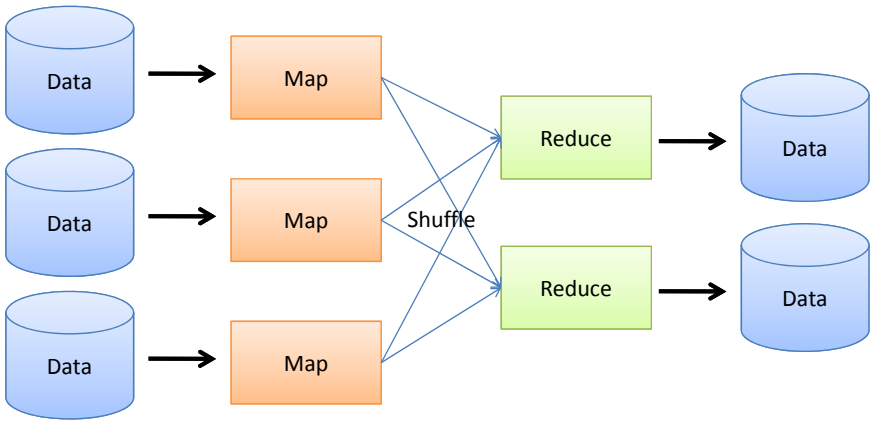
	Aug, '04	Mar, '05	Mar, '06
Number of jobs	29,423	72,229	171,834
Average completion time (secs)	634	934	874
Machine years used	217	981	2,002
Input data read (TB)	3,288	12,571	52,254
Intermediate data (TB)	758	2,756	6,743
Output data written (TB)	193	941	2,970
Average worker machines	157	232	268
Average worker deaths per job	1.2	1.9	5.0
Average map tasks per job	3,351	3,097	3,836
Average reduce tasks per job	55	144	147
Unique map/reduce combinations	426	411	2345

Google

MapReduce

Model

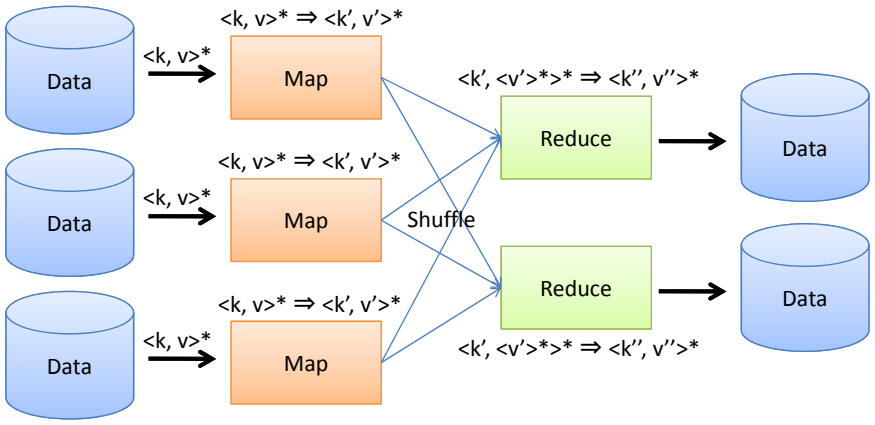
MapReduceの実行フロー



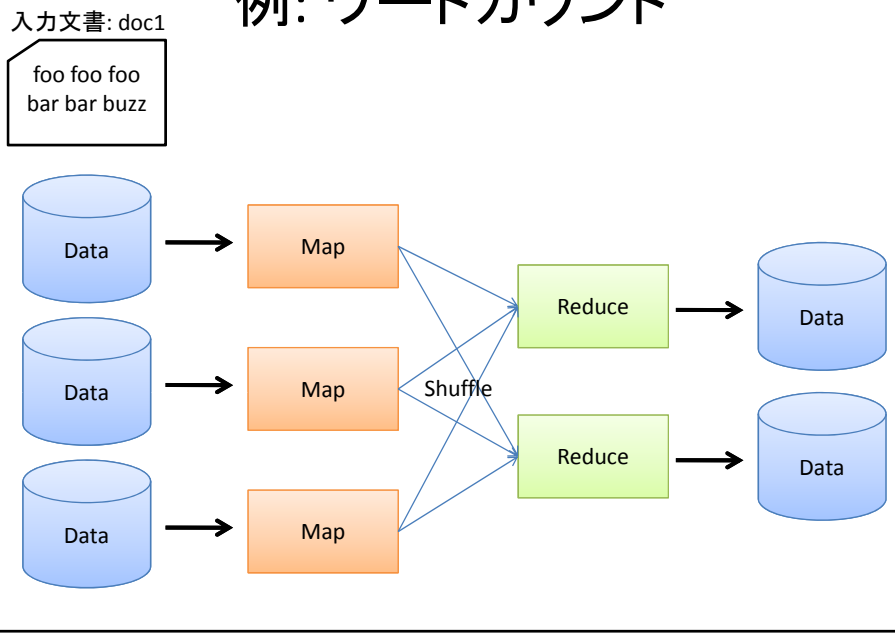
MapReduceの実行フロー

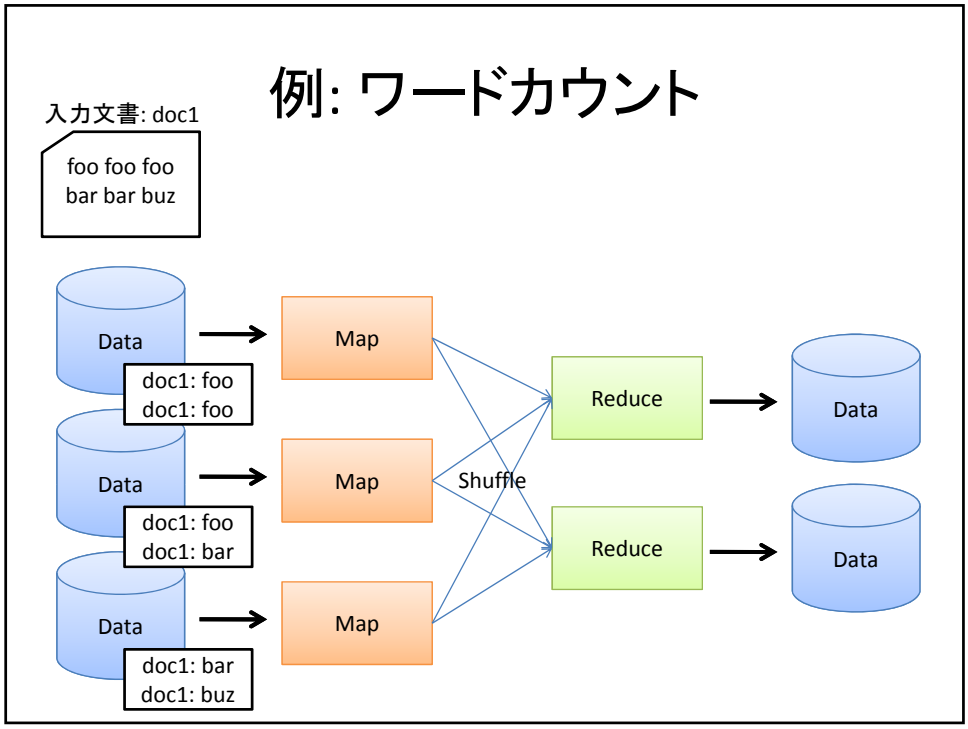
- 入力読み込み
  - <key, value>*
- Map
  - map: <key, value> ⇒ <key', value'>*
- Shuffle
  - shuffle: <key', reducers> ⇒ destination reducer
- Reduce
  - reduce: <key', <value'> * > ⇒ <key'', value''>*
- 出力書き出し
  - <key'', value''>*

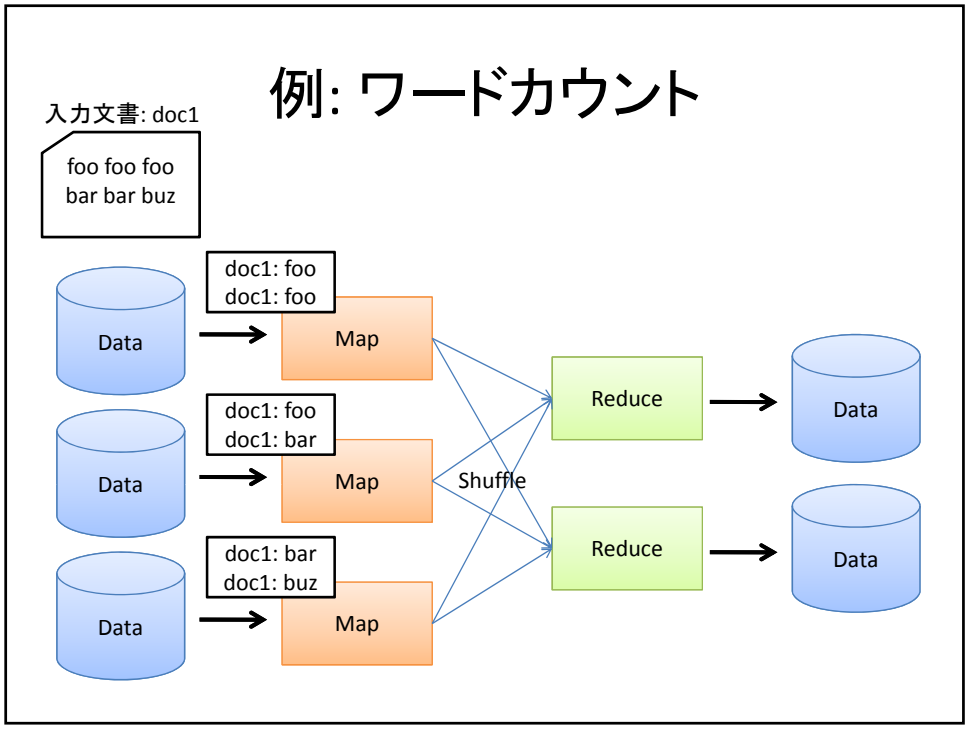
MapReduceの実行フロー

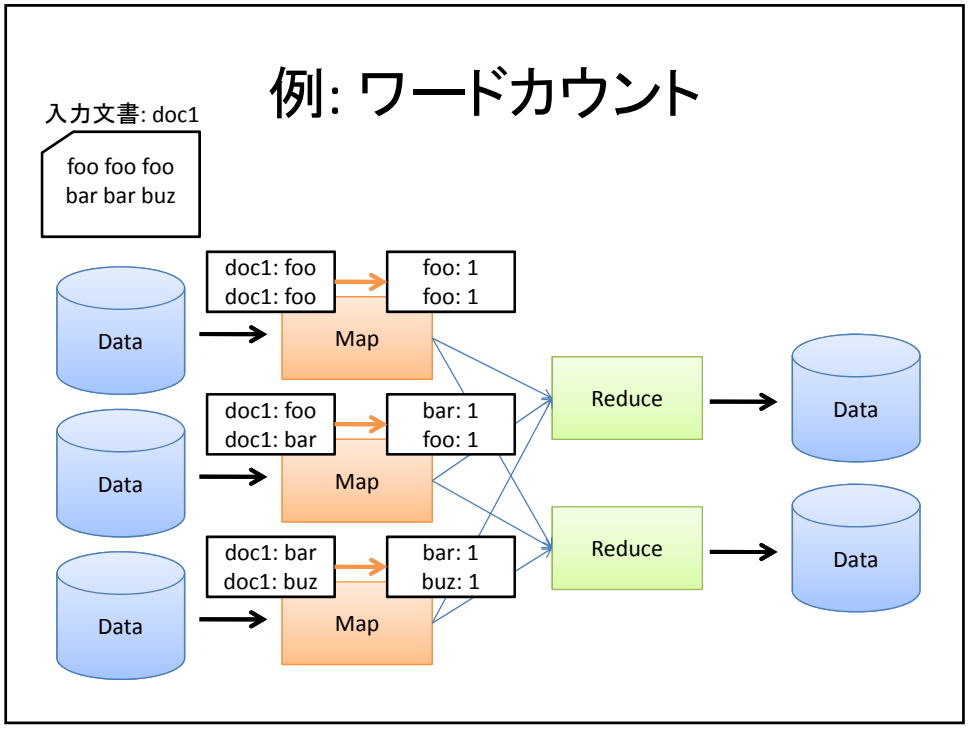


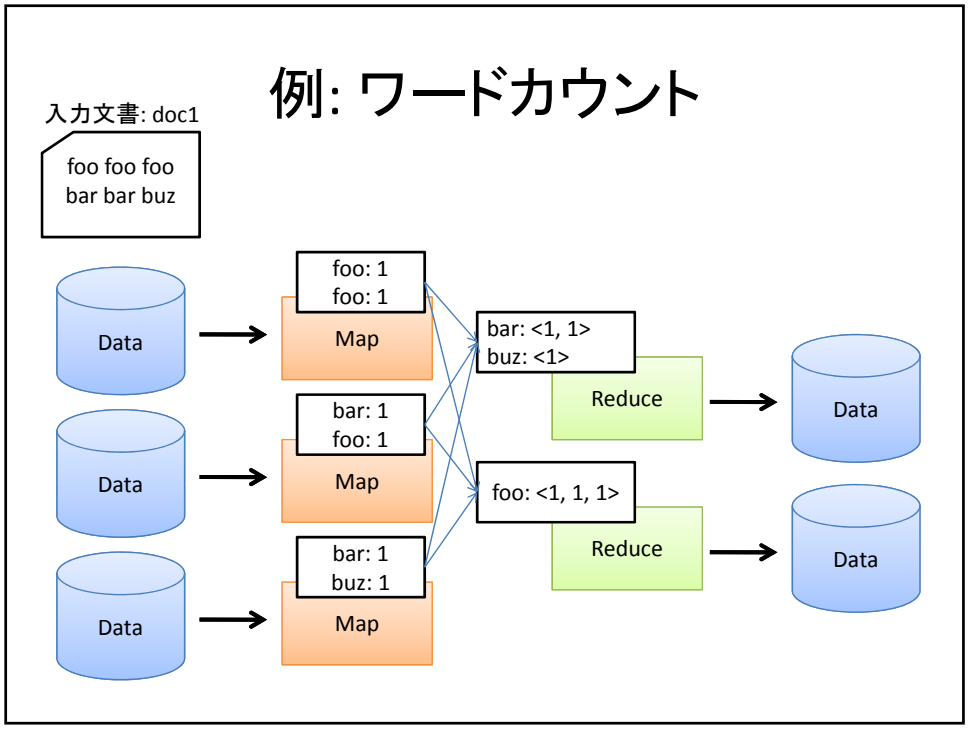
例: ワードカウント



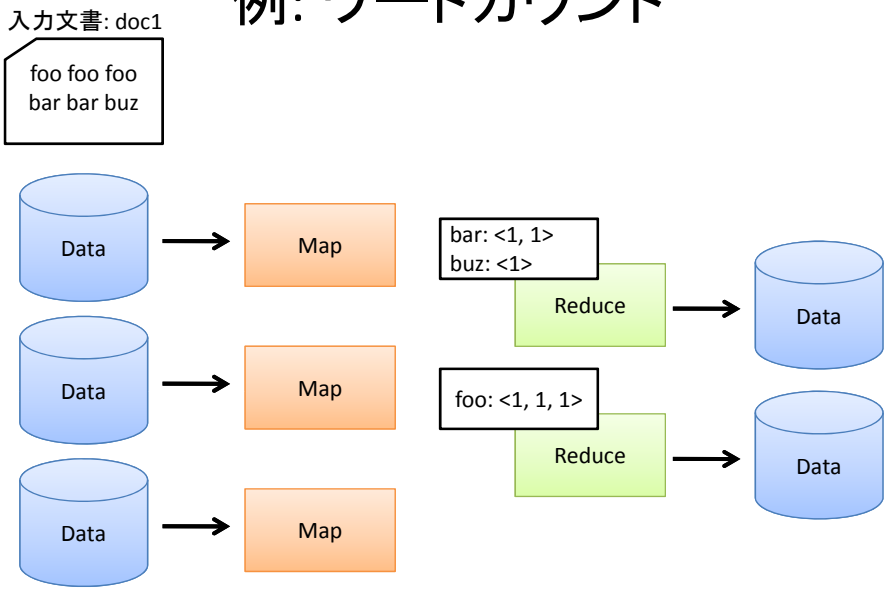

例: ワードカウント
入力文書: doc1
foo foo foo
bar bar buz
Data
Data
Data
doc1: foo
doc1: foo
doc1: foo
doc1: bar
doc1: bar
doc1: buz
Map
Map
Map
Shuffle
Reduce
Reduce
Data
Data


例: ワードカウント
入力文書: doc1
foo foo foo
bar bar buz
Data
Data
Data
doc1: foo
doc1: foo
doc1: foo
doc1: bar
doc1: bar
doc1: buz
Map
Map
Map
Shuffle
Reduce
Reduce
Data
Data

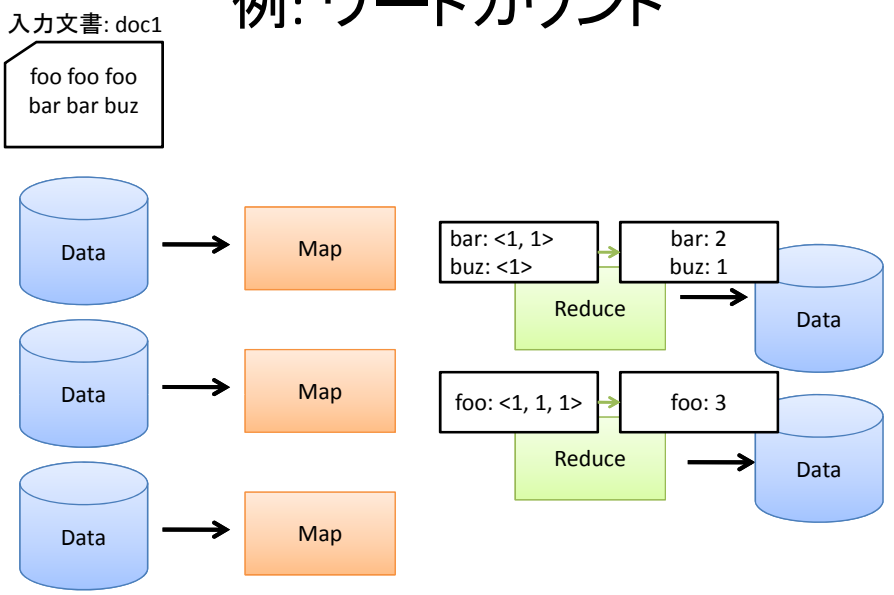

例: ワードカウント
入力文書: doc1
foo foo foo
bar bar buz
Data
Data
Data
doc1: foo
doc1: foo
foo: 1
foo: 1
Map
doc1: foo
doc1: bar
bar: 1
foo: 1
Map
doc1: bar
doc1: buz
bar: 1
buz: 1
Map
Reduce
Reduce
Data
Data


例: ワードカウント
入力文書: doc1
foo foo foo
bar bar buz
Data
Data
Data
foo: 1
foo: 1
Map
bar: 1
foo: 1
Map
bar: 1
buz: 1
Map
bar: <1, 1>
buz: <1>
foo: <1, 1, 1>
Reduce
Reduce
Data
Data

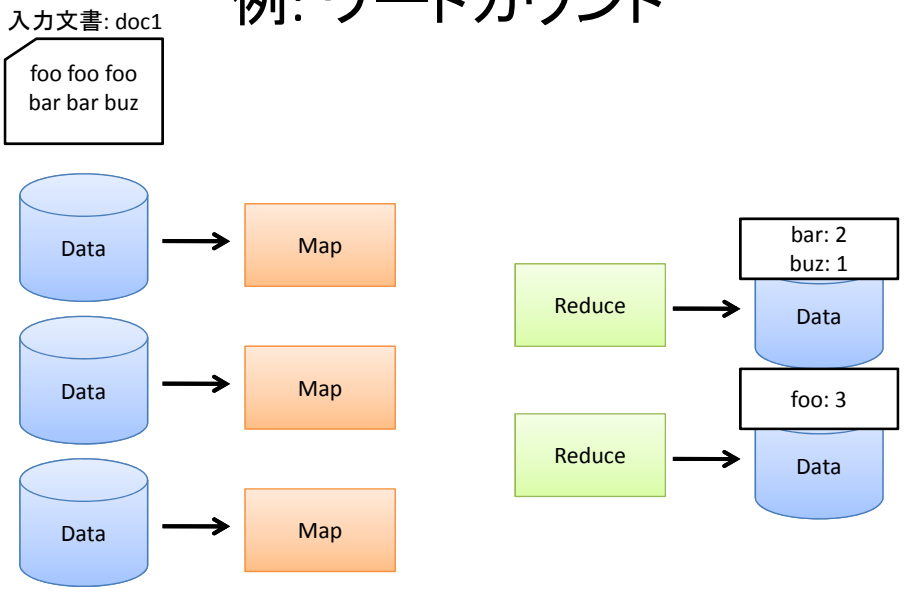
例: ワードカウント



例: ワードカウント



例: ワードカウント



例: ワードカウント

- 擬似コード

```
map(string key, string value) {
   foreach word in value:
      emit(word, 1);
}

reduce(string key, vector<int> values) {
   int result = 0;
   for (int i = 0; I < values.size(); i++)
      result += values[i];
   emit(key, result);
}
```

MapReduce型の処理

- Grep
- Sort (適切なPartition関数を選択する必要)
- Log Analysis
- Web Graph Generation
- Inverted Index Construction
- Machine Learning
  - NaiveBayes, K-means, Expectation Maximization, SVM, etc.

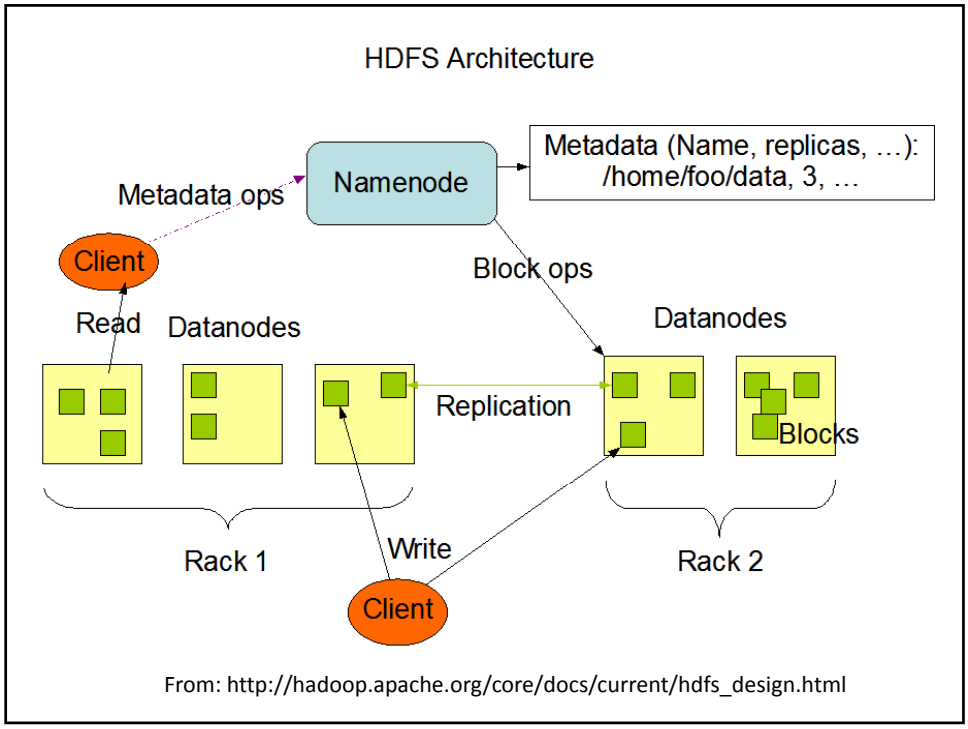
Hadoopの実装

Hadoopの中身

- Hadoop Distributed File System
  - GFSのクローン
  - MapReduceプログラムの入力や出力に使用
- MapReduce
  - MapReduceを実現するためのサーバー、ライブラリ等

Hadoop Distributed File System

- Master/Slave アーキテクチャ
- ファイルはブロックという単位に分割して保存
- NameNode
  - Master
  - ファイルのメタデータ(パス・権限など)を管理
- DataNode
  - Slave
  - 実際のデータ(ブロックを管理)

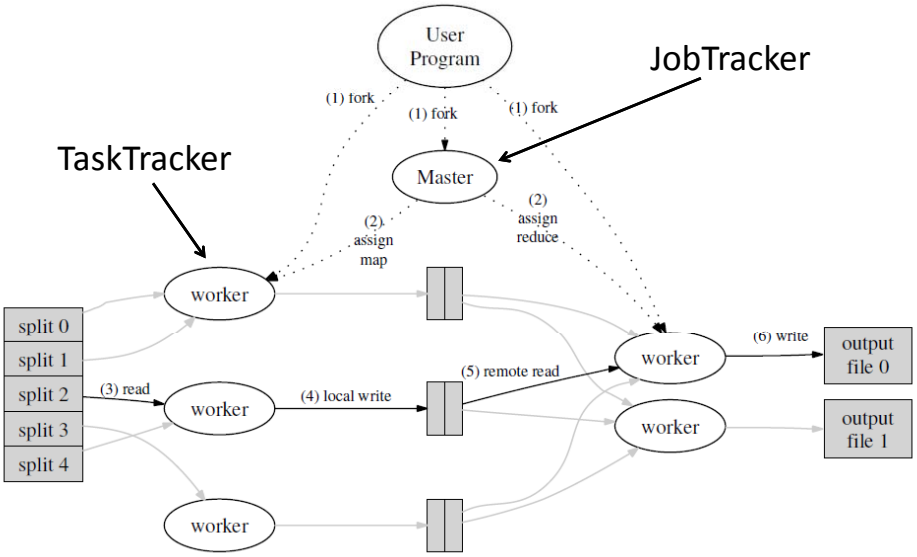


From: http://hadoop.apache.org/core/docs/current/hdfs_design.html

Hadoop MapReduce

- Master/Slave アーキテクチャ
- JobTracker
  - Master
  - JobをTaskに分割し、Taskを各TaskTrackerに分配
    - Job: MapReduceプログラムの実行単位
    - Task: MapTask, ReduceTask
  - 全てのTaskの進行状況を監視し、死んだり遅れたりしたTaskは別のTaskTrackerで実行させる
- TaskTracker
  - Slave
  - JobTrackerにアサインされたTaskを実行
    - 実際の計算処理を行う

MapReduce Architecture



HadopStreamng
Rubyによるワードカウント

```
$ ./bin/hadoop
  jar contrib/hadoop-0.15.3-streaming.jar
  -input wcinput
  -output wcoutput
  -mapper /home/hadoop/kzk/map.rb
  -reducer /home/hadoop/kzk/reduce.rb
  -inputformat TextInputFormat
  -outputformat TextOutputFormat
```

map.rb

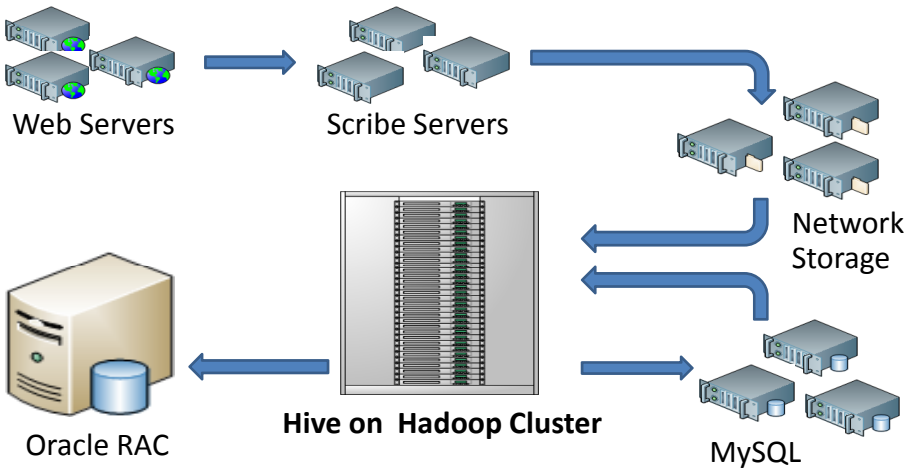
```
#!/usr/bin/env ruby
while !STDIN.eof?
  line = STDIN.readline.strip
  ws = line.split
  ws.each { |w| puts "#{w}¥t1“ }
end
```

reduce.rb

```
#!/usr/bin/env ruby
h = {}
while !STDIN.eof?
  line = STDIN.readline.strip
  word = line.split("¥t")[0]
  unless h.has_key? word
    h[word] = 1
  else
    h[word] += 1
  end
end
h.each { |w, c| puts "#{w}¥t#{c}“ }
```

Facebookのログ処理事例
Scribe + Hive

Facebook Architecture



Scribe

- Facebookの分散ログ収集ソフト
  - http://sourceforge.net/projects/scribeserver/
  - 10/27にOSS化
- 各サーバーはログを単に中継する役割
  - 設定ファイルベースでトポロジーを設定
  - 中継先が落ちている場合はディスクに書き込み
    - ただし、毎回のsyncはしない
    - 多少は失われるけど、大丈夫だよね
- Thriftを使用
  - 様々な言語のプログラムから利用可能

Facebook Hive

- Hadoop上に構築されたデータ処理基盤
  - http://wiki.apache.org/hadoop/Hive
  - struct, list, map等の構造化されたデータをSQLライクな言語(HiveQL)で処理可能
  - HDFS, Hadoop MapReduceを大いに活用
- Hadoopのcontribにコミット済み
  - FBでは320台、2560 core、1.3PBで運用
  - レポーティング、機械学習に使用

HiveQL Example: Join Operation

page_view

pageid	**userid**	time
1	**111**	9:08:01
2	**111**	9:08:13
1	**222**	9:08:14

X

user

userid	Age	gender
111	25	female
222	32	male

=

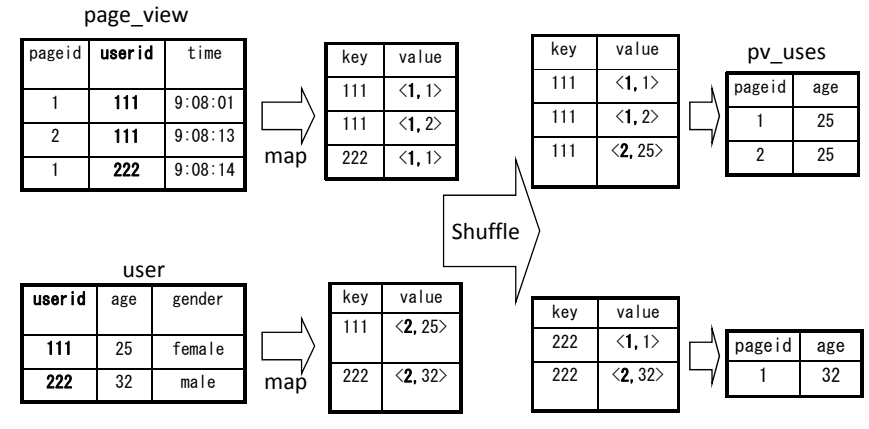
pv_users

pageid	age
1	25
2	25
1	32

- SQL:

```
INSERT INTO TABLE pv_users
SELECT pv.pageid, u.age
FROM page_view pv JOIN user u ON (pv.userid = u.userid);
```

HiveQL Join by MapReduce




```
INSERT INTO TABLE pv_users
SELECT pv.pageid, u.age
FROM page_view pv JOIN user u ON (pv.userid = u.userid);
```

HiveQL: 他の操作

- GROUPBY, DISTINCT JOIN
  - MapReduceで実装可能
- 任意のMR用スクリプトを簡単に使用可能
  - FROM (
    - FROM pv_users
    - SELECT **TRANSFORM**(pv_users.userid, pv_users.date)
    - **USING** 'map_script' AS (dt, uid)
    - **CLUSTER BY** dt) map

Enjoy Playing Around Hadoop ☺

Thank you!

kzk < kzk@preferred.jp>