入門

Kazuki Ohta
(株) Preferred Infrastructure
最高技術責任者

<kzk@preferred.jp>

自己紹介

「Hadoop徹底入門」

- 太田一樹
  - (株) Preferred Infrastructure, CTO
  - エンタープライズ検索エンジン「Sedue」
  - Hadoopユーザー会の立ち上げ
  - Hadoop徹底入門の著者
- 個人サイト
  - http://kzk9.net/
  - @kzk_mover

Hadoop-Gfarm with 三上さん

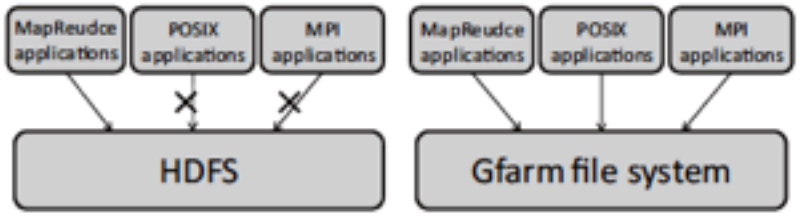


図 1 HDFS 上では Haodop MapReduce アプリケーションしか実行出来ないが、Gfarm 上では通常の POSIX アプリケーションや MPI のアプリケーションを実行できる

- 東京大学情報科学科 石川研究室 修士卒業
  - 並列I/Oシステムの研究
  - Project: IOFSL (I/O Forwarding and Scalability Layer)

Preferred Infrastructure

Agenda

- 大規模データ処理とその課題
- Hadoopとは?
  - ソフトウェア構成
  - 内部アーキテクチャ
  - エコシステム
- MapReduce入門
  - MapReduceの背景
  - MapReduceの計算モデル

大規模データ処理とその課題

大規模データ処理とは?

- 「大量のデータを処理し、
  その中から知見を抽出すること」
  - データを価値に変える

- 例1: バスケット分析
  - 大量のPOSデータを分析し、商品Aと同時によく買われている商品を分析する (例: ビールとおむつ)
  - その2つを近くに陳列する事で、収益の向上が見込める

- 例2: 交通流量分析
  - 自動車には位置情報を収集できるセンサーが搭載されており、そのデータを解析することで、頻繁に渋滞が起こる箇所の特定が可能
  - 道路の増強を行なうことで、渋滞改善が見込める

Preferred Infrastructure

大規模データ処理の抱える問題点

- 大量のデータを保存・処理する際の問題
- 問題1: データの保存
  - 数百TB ～ 数PBレベルのデータを、安全に保存する必要が有る
  - 1つのディスクに収まらないので、大量のディスクを使用する
    - 数十～数千ノード程度のクラスタシステム => 故障率の大幅増加
  - ディスクの故障が起きても、データが失われないような設計が必要
- 問題2: データの処理
  - 保存されたデータに対して処理を行なう必要が有る
  - 処理プログラムの開発の効率性・再利用性・デバッグ環境
  - データ処理中のノード故障にも対処する必要が有る

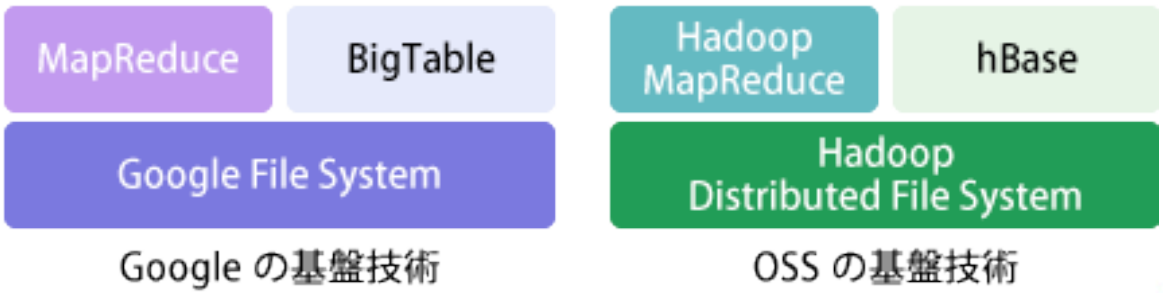
Preferred Infrastructure

Hadoopとは?

Hadoopとは?

- Google社が保有する大量データ処理基盤技術のOSSクローン
  - Apacheプロジェクトにて全ソースコードが公開

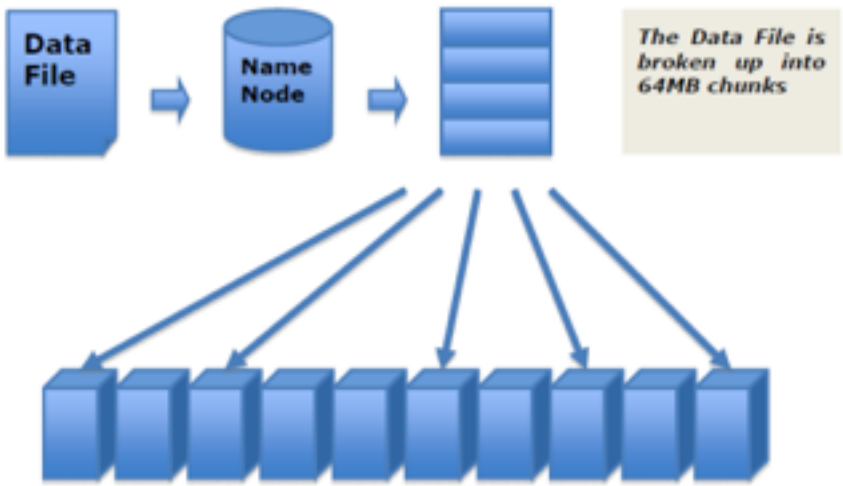
- 大量データの保存技術 “Google File System”
- 大量データの処理技術 “MapReduce”
  - これらの技術を、学術論文を元に再実装
  - Java言語で記述されており、Yahoo!では4000台での稼働実績有
  - 本日は “MapReduce” を中心に解説します



Preferred Infrastructure

Hadoop Distributed File Systemとは?

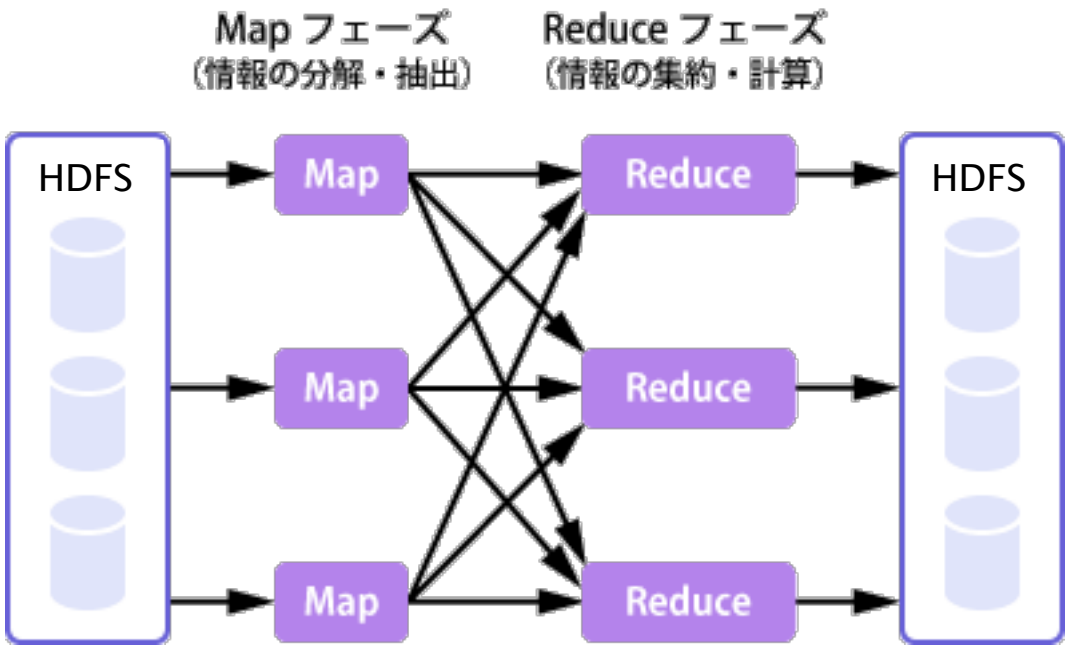
- 大量のコモディティマシンを使用し、大規模データを安全に保存するためのソフトウェア
  - データは自動的に複数箇所にコピーされ、故障時にもデータが失われる確率が低い (例: 複数ラック, 複数DC)
  - マスターノードが、大量のスレーブノードを管理・監視



Preferred Infrastructure

MapReduceとは?

- HDFS上に構築された、大量データ処理基盤
  - Mapフェーズ, Reduceフェーズの処理を記述するだけで、クラスタ環境での並列処理可能
  - 処理中のマシンの故障等も自動的にフェイルオーバー



Preferred Infrastructure

hadoop の開発者

- Cloudera (元: Yahoo Research) の Doug Cutting氏が開発
  - 元々はLucene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から命名
- 現在は、Yahoo!を中心にFacebook, Cloudera等の企業の従業員が主に開発に従事している

Hadoopの使用事例

- Webログ解析
  - ユニークユーザー数算出
  - 広告ターゲティング
- マーケティングデータ解析
  - Twitter・Blog解析
- POSデータ解析
  - 時系列バスケット分析
- センサデータ解析
  - 異常値検出・位置情報分析
- 機械学習処理
  - ベイズ推定
  - クラスタリング
  - 回帰分析
- ゲノム解析処理
- 金融取引データ処理
- スマートグリッド
- …

- 国外では Yahoo!, Facebook, eBay, Amazon等、多くの企業が活用
- 国内でも 楽天・クックパッド・リクルート・NTTデータ等が活用

Preferred Infrastructure

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

MapReduceに関する学会

- 分散システム, HPC, 自然言語処理, 機械学習, etc.
- MAPREDUCE'10
  - The First International Workshop on MapReduce and its Applications
  - HPDC 2010併設
  - http://graal.ens-lyon.fr/mapreduce/
- MAPRED'2010
  - The First Internatinal Workshop on Theory and Practice of MapReduce
  - CloudCom 2010併設
  - http://salsahpc.indiana.edu/CloudCom2010/mapreduce2010.html

Preferred Infrastructure.

MapReduce を利用している分野

- 広告解析
- バイオインフォマティクス / 医療情報
- 機械翻訳
- 地理情報処理
- 情報抽出、テキスト解析
- 機械学習 / データマイニング
- 検索クエリ分析
- 情報検索
- スパム ・マルウェア判定
- 画像・動画処理
- ネットワーク処理
- シミュレーション
- 統計処理
- 数値解析
- グラフアルゴリズム

http://atbrox.com/2010/05/08/mapreduce-hadoop-algorithms-in-academic-papers-may-2010-update/

その他: Analyzing Human Genomes with hadoop

http://www.cloudera.com/blog/2009/10/analyzing-human-genomes-with-hadoop/

Preferred Infrastructure.

MapReduceの主な適用可能分野

1. Modeling true risk
2. Customer churn analysis
3. Recommendation engine
4. Ad targeting
5. PoS transaction analysis
6. Analyzing network data to predict failure
7. Threat analysis
8. Trade surveillance
9. Search quality
10. Data “sandbox”

Ten Hadoopable Problems

http://www.slideshare.net/cloudera/20100806-cloudera-10-hadoopable-problems-webinar-4931616

Preferred Infrastructure.

Yahoo!の使用事例

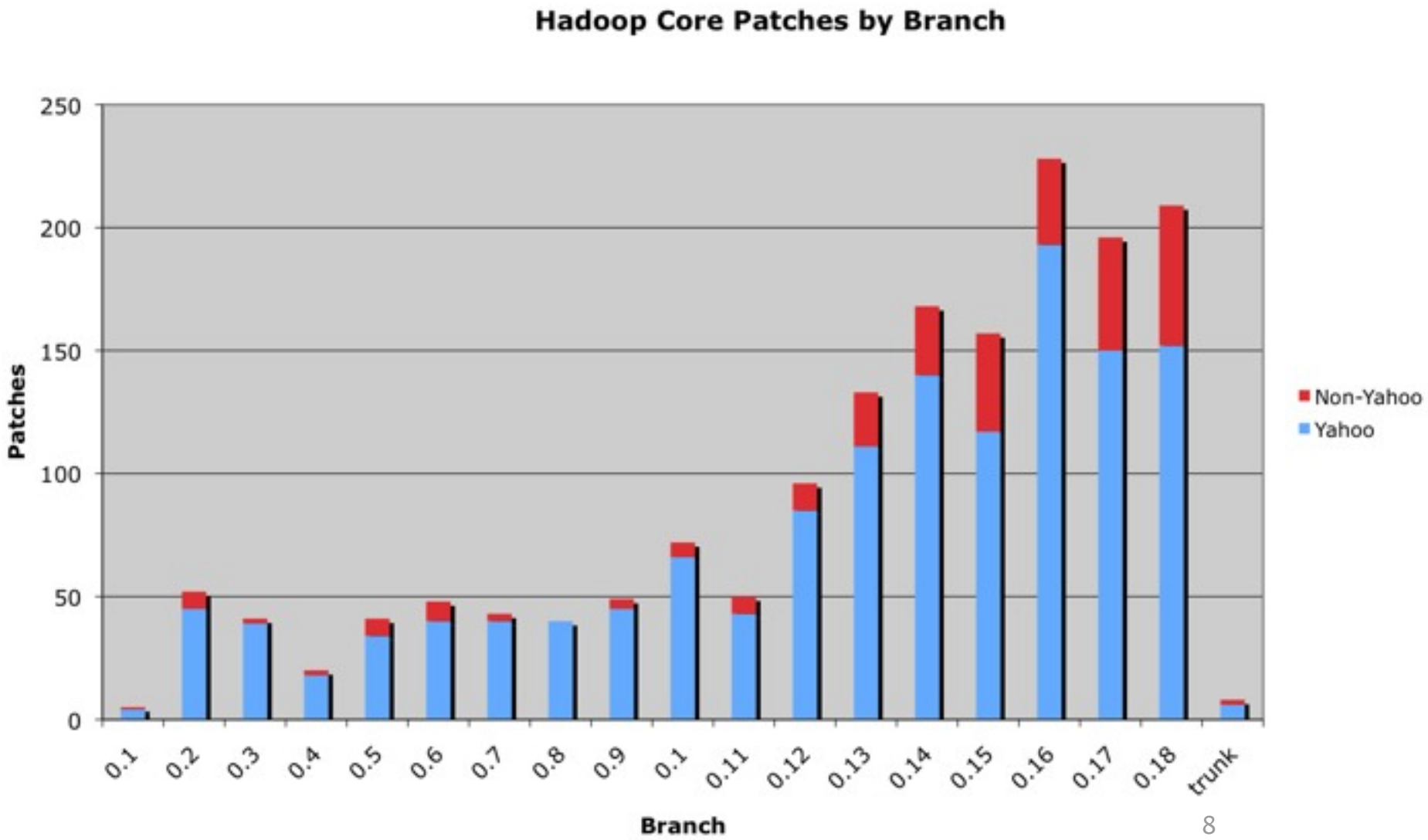
- 約25000ノード
  - 20%が本番用、60%が研究用
  - 広告最適化、検索インデックス作成、RSSフィード、スパムフィルタ、パーソナライズ化
- ログデータの分析時間の短縮
  - 過去3年分の解析が 26日 -> 20分
- 分析アプリケーションの開発時間の短縮
  - C++ からPythonに代わり 2-3週 -> 2-3日

http://www.publickey1.jp/blog/09/hadoophadoop_worldny_2009.html

Hadoopに関わる企業

- Cloudera
  - HadoopをEnterprise向けに提供する企業
  - 多数のコミッターを抱える
  - http://cloudera.com/
- Yahoo!, Inc.
  - 検索エンジンバックエンド等にHadoopを使用
  - 米国で最大級の使用事例
  - Pig, Oozie等の周辺ソフトウェアも開発
- Facebook
  - ログ解析プラットホーム等にHadoopを使用
  - Hive等の周辺ソフトウェアも開発

Hadoopの開発状況



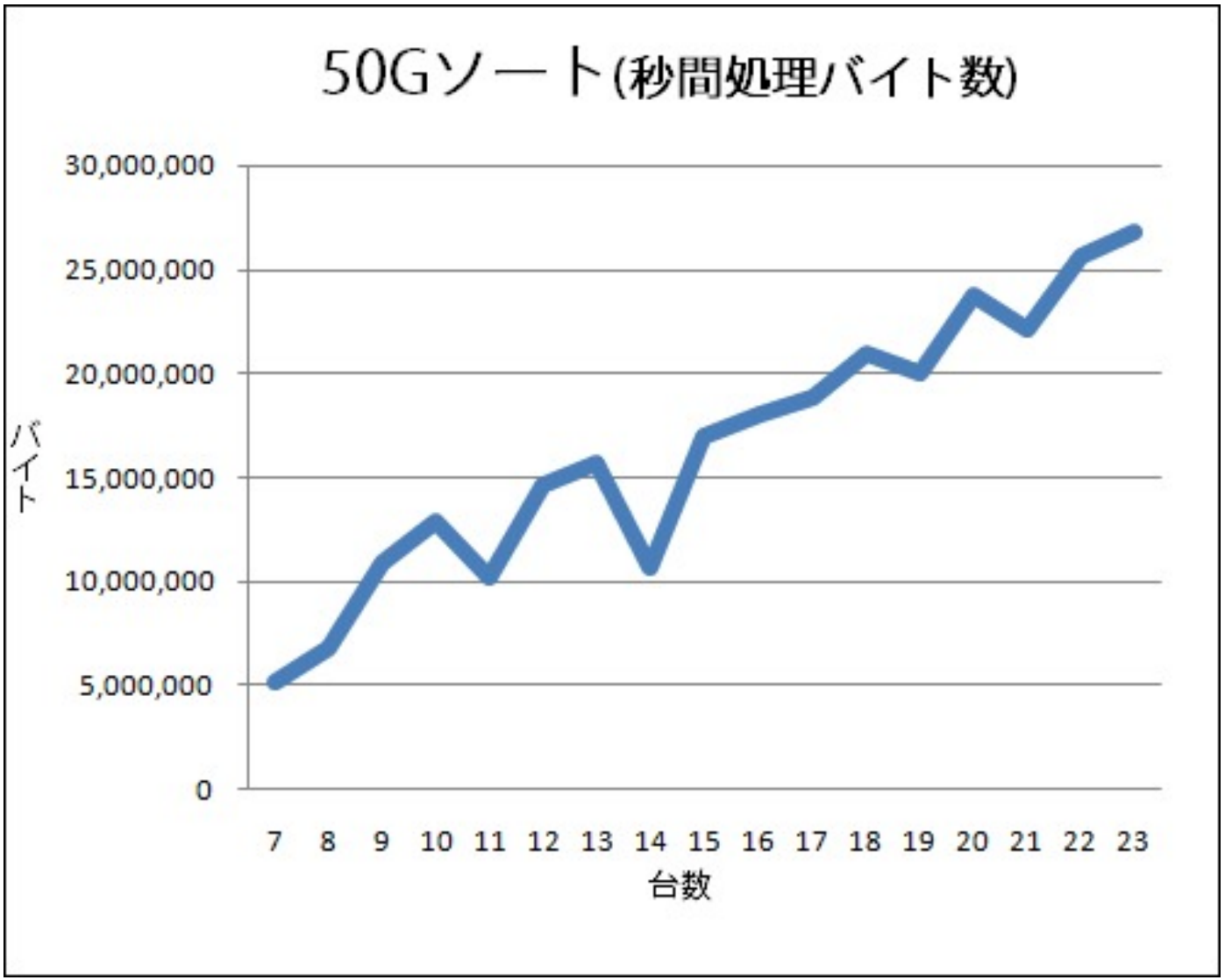
Hadoop参考文献

- 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, 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
- オープンソース分散システム「Hadoop」解析資料
  - http://preferred.jp/2008/08/hadoop.html
- Hadoopユーザー会メーリングリスト
  - http://groups.google.co.jp/group/hadoop-jp

性能

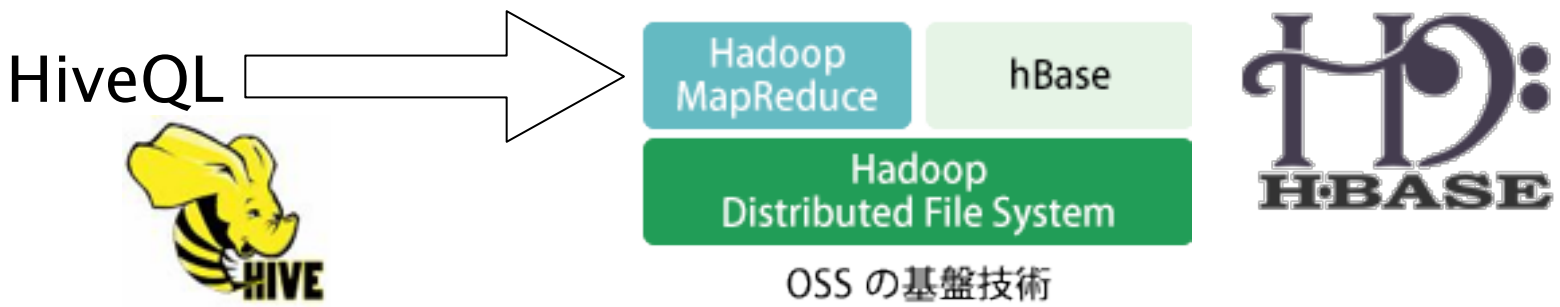
- Apache Hadoop wins TeraSort Benchmark
  - http://sortbenchmark.org/
  - 規定フォーマットの100Tデータをソート
  - Yahoo!のチームによるレポート
    - http://sortbenchmark.org/Yahoo2009.pdf
  - 173 minutes, 3452 nodes
    - 40 nodes per rack
    - 2 quad core Xeon
    - 8 GB Memory
    - 4 Sata Disks
    - 1Gbps Ethernet, 8Gbps uplinks per rack
    - 秒間約570MB/sec

性能測定


50Gソート(秒間処理バイト数)
30,000,000
25,000,000
20,000,000
15,000,000
10,000,000
5,000,000
0
バイト
7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23
台数

Hadoopのエコシステム

- Hadoopを軸に、大小様々なソフトウェアが開発されている
- Hive
  - SQLライクな言語で大量データの解析が可能
  - SELECT user, COUNT(1)
    FROM log_tbl
    WHERE user.sex == “male”
    GROUP BY user
- HBase
  - HDFS上に構築された分散データベース
  - リアルタイム分析処理を可能にするミドルウェア



Preferred Infrastructure

MapReduceとは?

問題

- Web、大規模インターネットサイト、モバイルキャリア等では、非常に大規模なデータが蓄積されている
  - 例えばWebページのサイズを考えてみる
    - 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500Gのディスクが1000個程度必要
- 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
    - SPMD
  - 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プログラムの例

```
 1 : #include <stdio.h>
 2 : #include <stdlib.h>
 3 : #include <math.h>
 4 :
 5 : #include "mpi.h"
 6 :
 7 : int main(int argc, char *argv[])
 8 : {
 9 :     int  namelen, num_procs, myrank;
10 :     char processor_name[MPI_MAX_PROCESSOR_NAME];
11 :
12 :     MPI_Init(&argc, &argv);
13 :
14 :     MPI_Comm_size(MPI_COMM_WORLD, &num_procs);
15 :     MPI_Comm_rank(MPI_COMM_WORLD, &myrank);
16 :     MPI_Get_processor_name(processor_name, &namelen);
17 :
18 :     printf("Hellow, MPI! at Process %d of %d on %s\n", myrank, nu
   m_procs, processor_name);
19 :
20 :     MPI_Finalize();
21 :
22 :     return EXIT_SUCCESS;
23 : }
24 :
```

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必要があり、非常に面倒くさい
  - 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

そこでMapReduce

- 大体の大規模データ処理を行う問題に特化したプログラミングモデル
  - アルゴリズムの記述のみにプログラマが集中できる
  - 世の中の問題全てに対して最適なモデルではない
- ライブラリ側で面倒な事を全て担当してくれる
  - 自動的に処理を分散/並列化
  - ロードバランシング
  - ネットワーク転送・ディスク使用効率化
  - 耐障害性の考慮
    - 1ノードで失敗したら違うノードで自動的に再実行
  - MapReduceが賢くなれば、それを使う全てのプログラムが賢くなる

MapReduce型の処理

- WordCount
- Grep
- Sort
- Log Analysis
- Web Graph Generation
- Inverted Index Construction
- Machine Learning
  - NaiveBayes, K-means, Expectation Maximization, etc.

Preferred Infrastructure.

MapReduce を利用している分野

- 広告解析
- バイオインフォマティクス / 医療情報
- 機械翻訳
- 地理情報処理
- 情報抽出、テキスト解析
- 機械学習 / データマイニング
- 検索クエリ分析
- 情報検索
- スパム ・マルウェア判定
- 画像・動画処理
- ネットワーク処理
- シミュレーション
- 統計処理
- 数値解析
- グラフアルゴリズム

大量データに対するバッチ処理にマッチしやすい

http://atbrox.com/2010/05/08/mapreduce-hadoop-algorithms-in-academic-papers-may-2010-update/

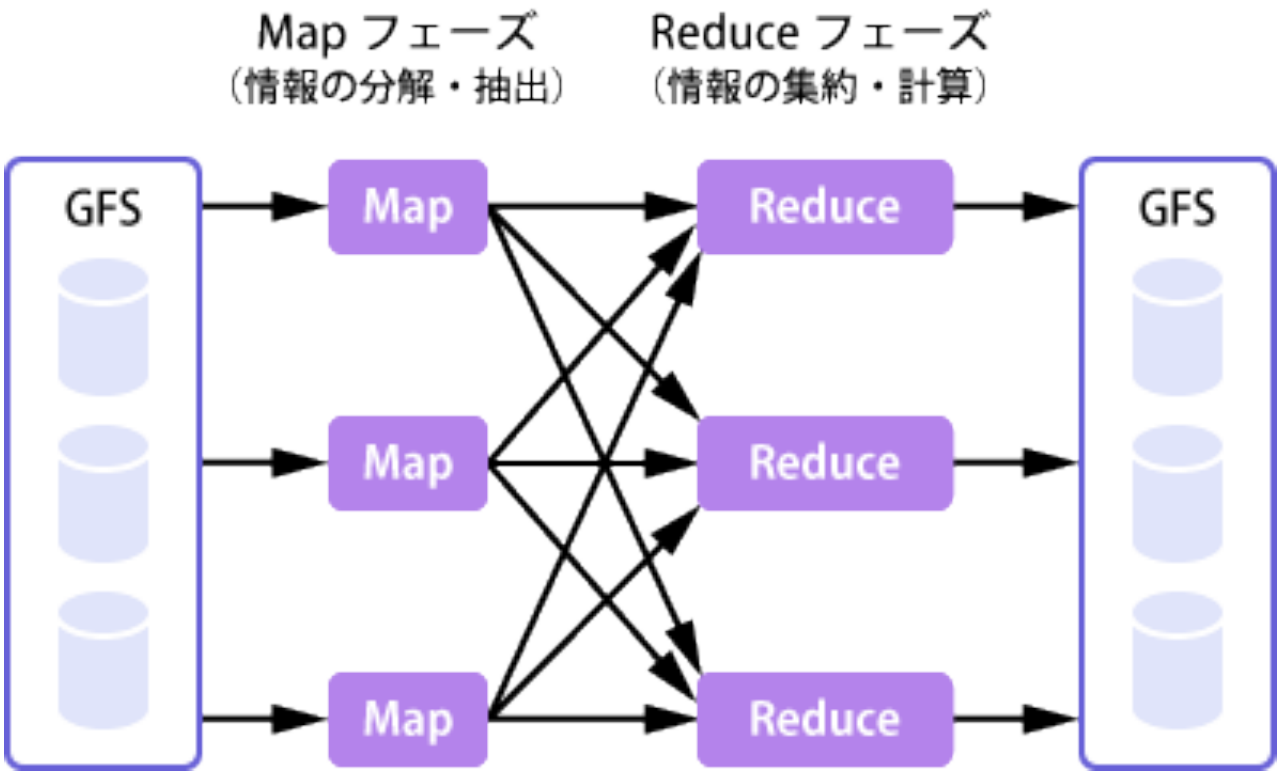
Googleでの使用率

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

MapReduceの計算モデル

MapReduceの実行フロー



MapReduceの実行フロー

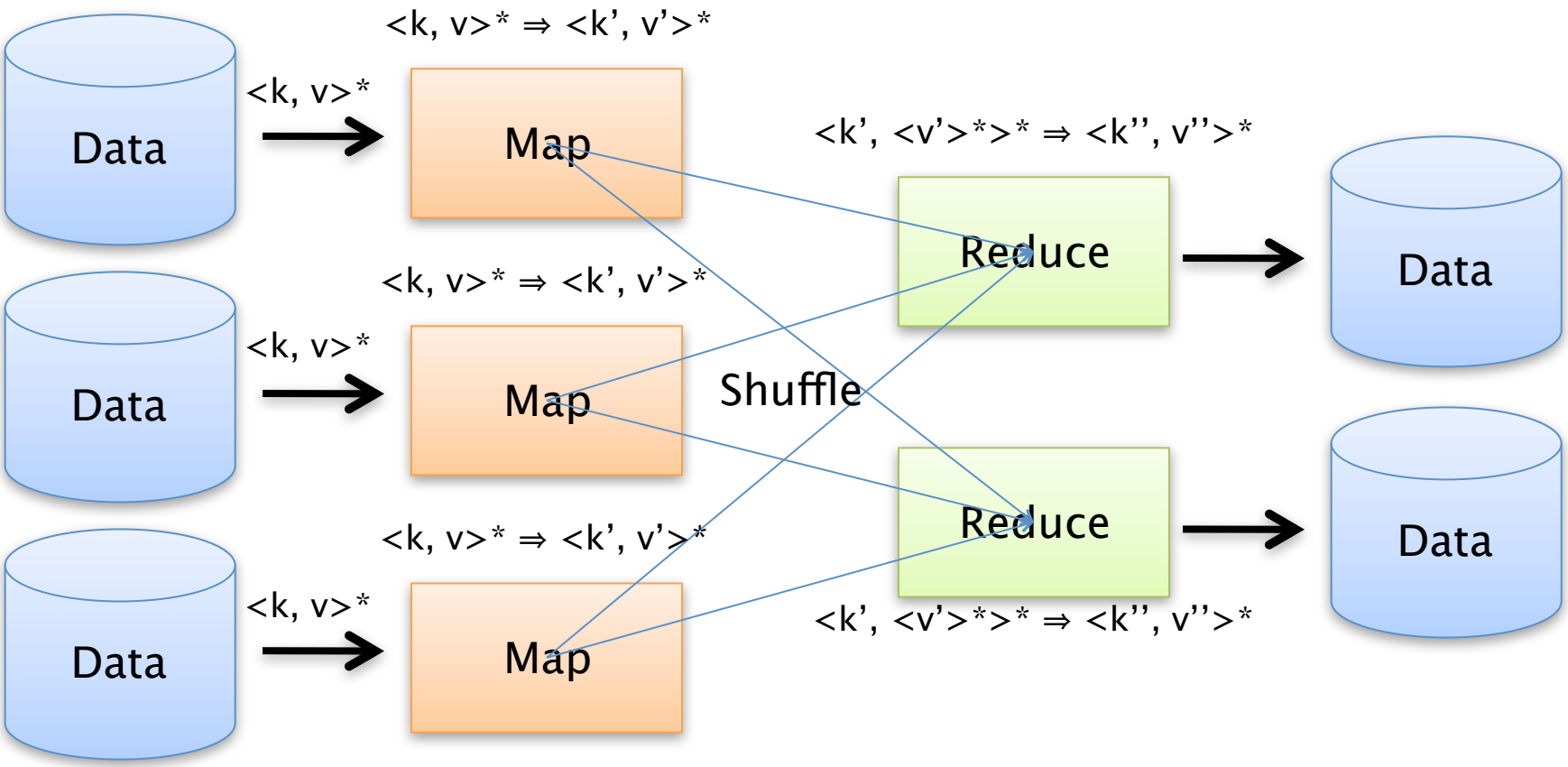
- 入力読み込み
  - <key, value>*
- Map
  - map: <key, value> ⇒ <key', value'>*
- Shuffle
  - shuffle: <key', reducers> ⇒ destination reducer
- Reduce
  - reduce: <key', <value'> * > ⇒ <key'', value''>*
- 出力書き出し
  - <key'', value''>*

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);
}
```

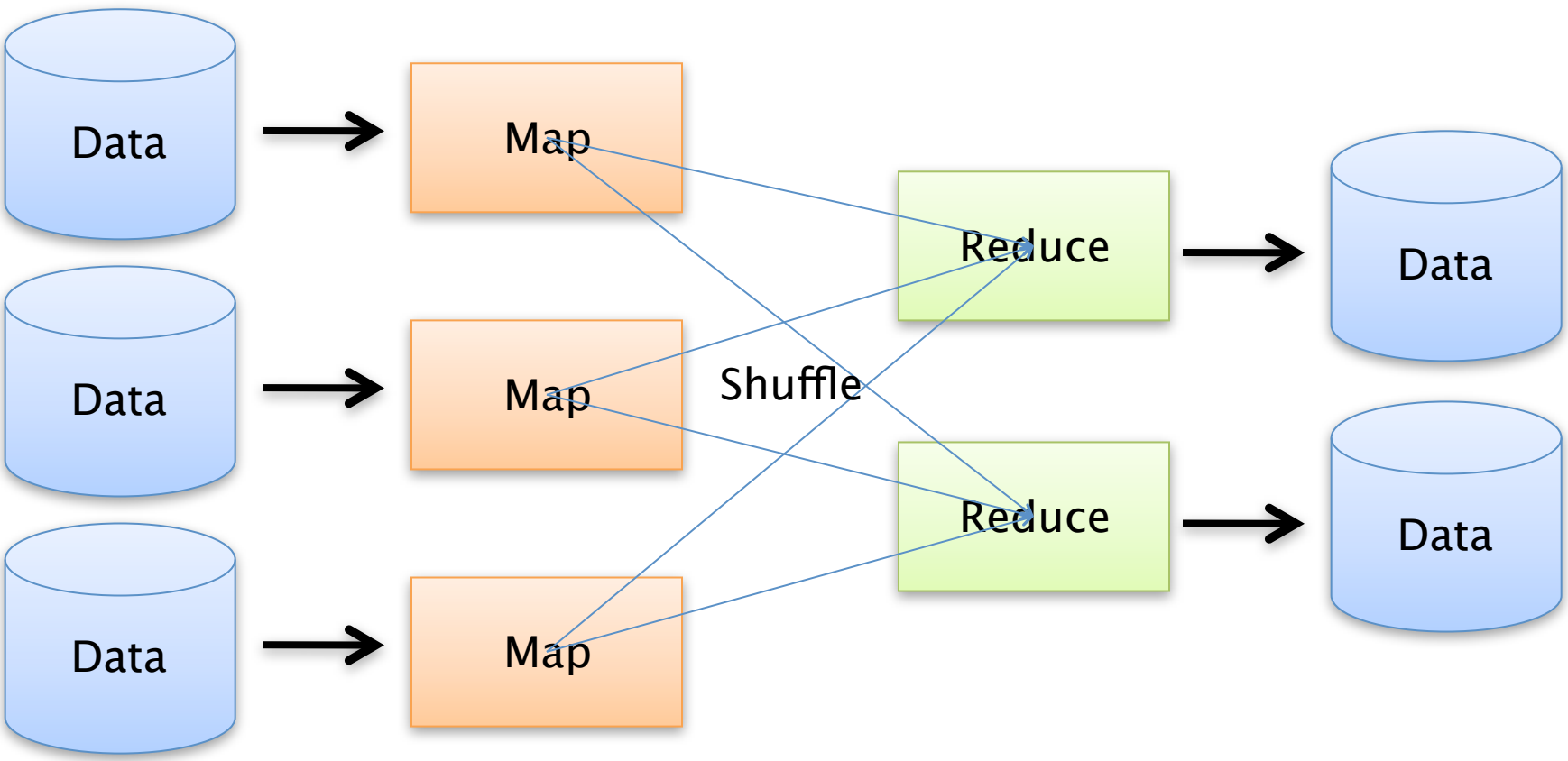
MapReduceの実行フロー



例: ワードカウント

入力文書: doc1

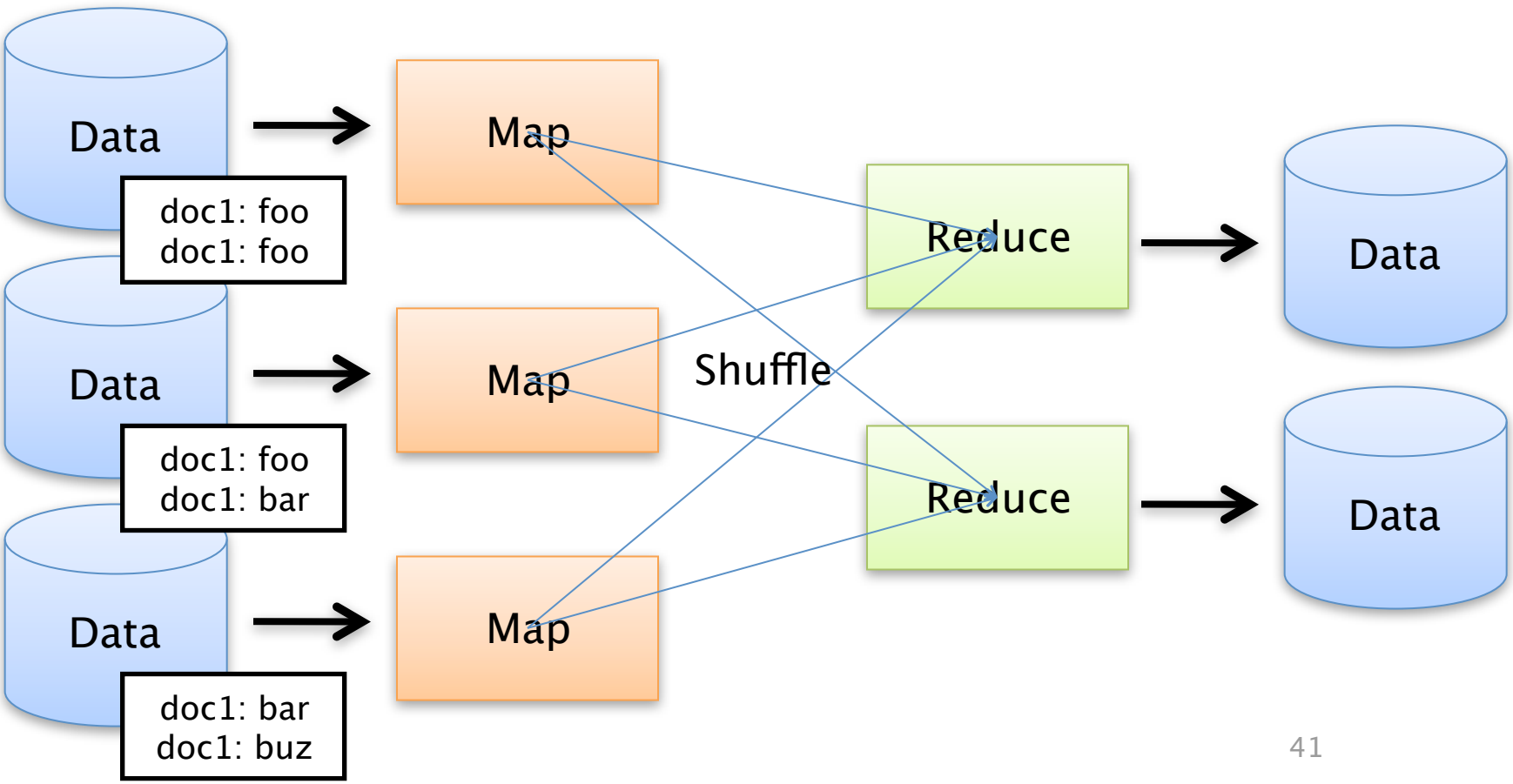
foo foo foo
bar bar buzz



例: ワードカウント

入力文書: doc1

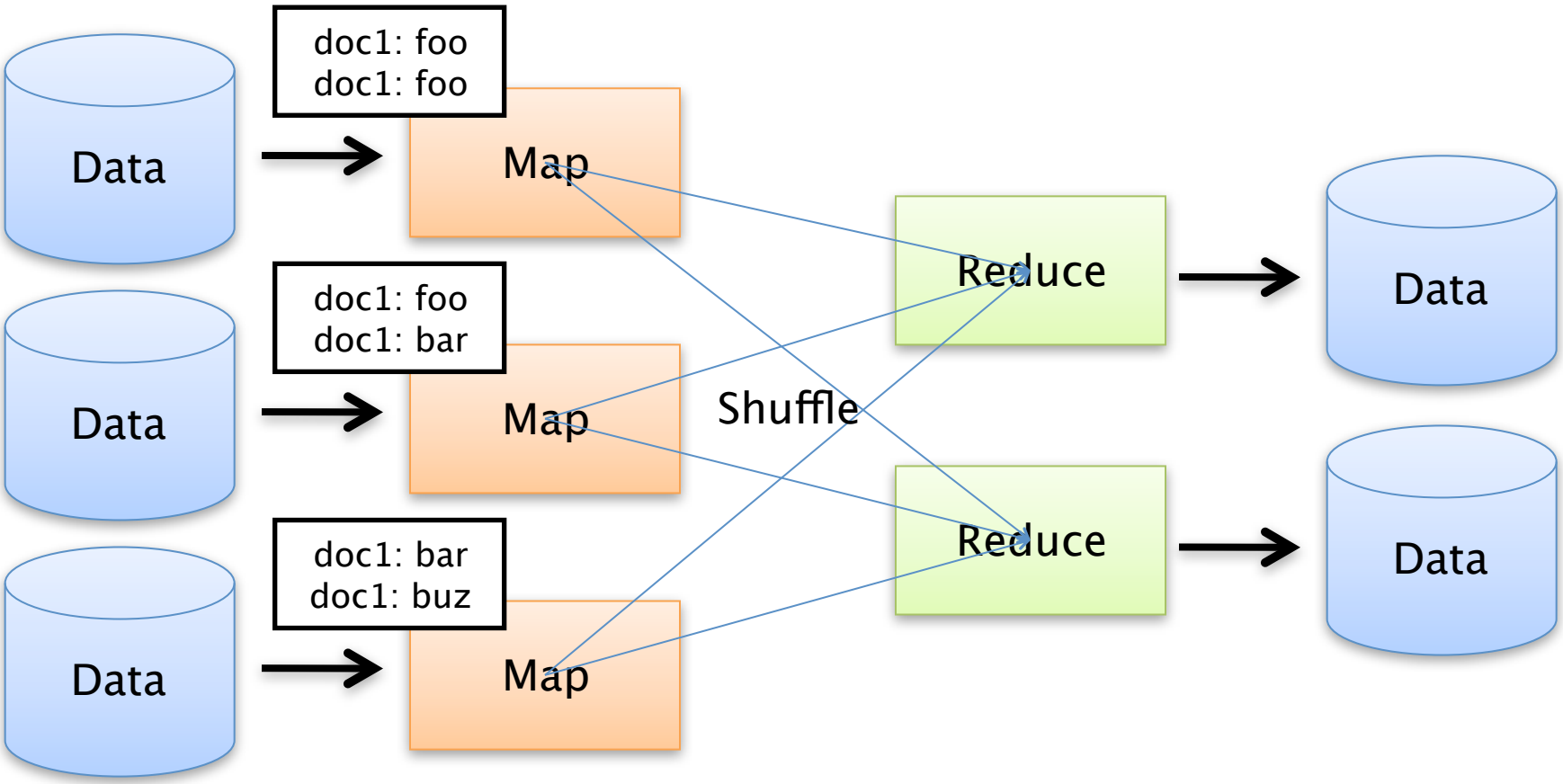
foo foo foo
bar bar buz



例: ワードカウント

入力文書: doc1

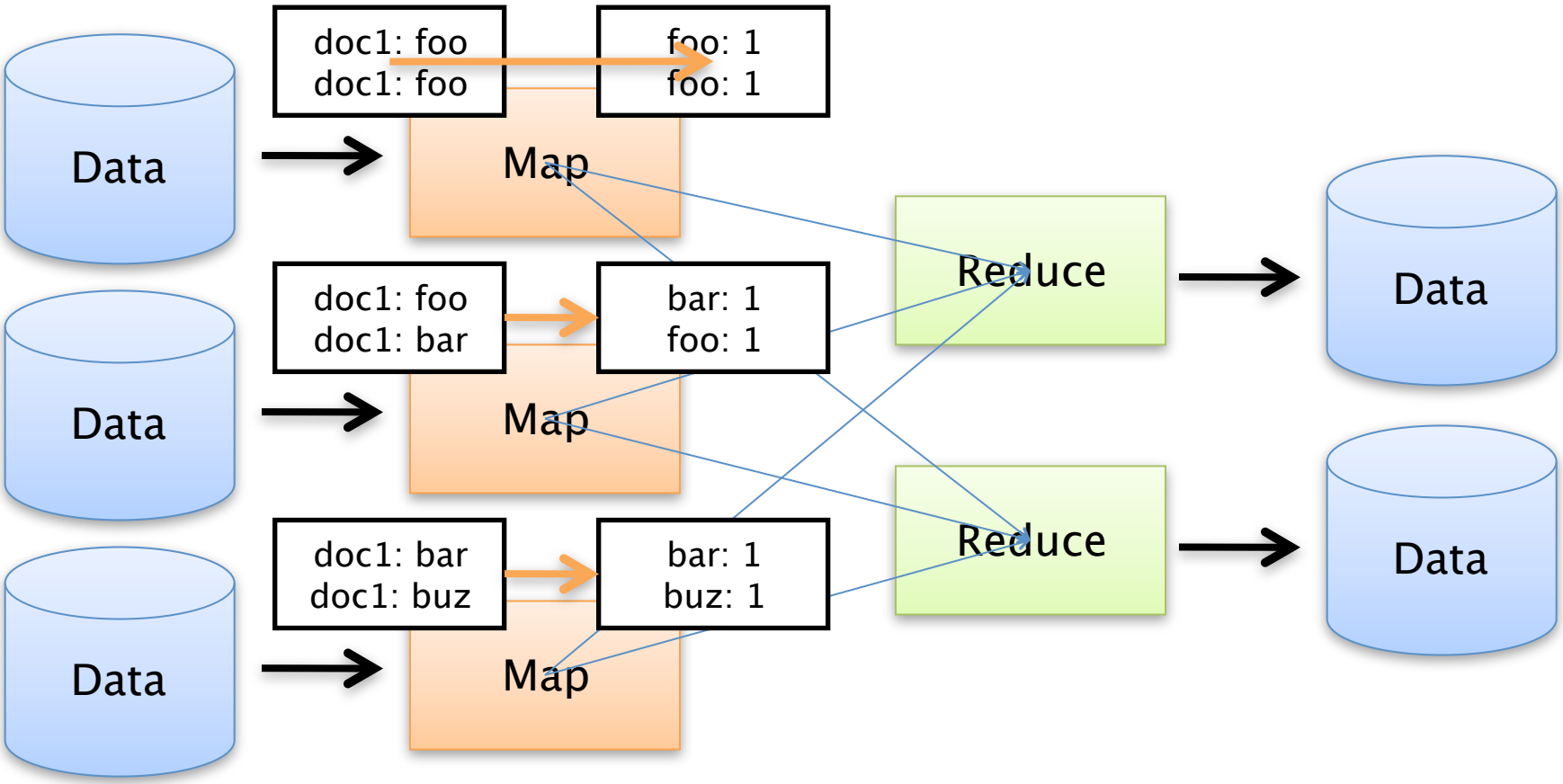
foo foo foo
bar bar buz



例: ワードカウント

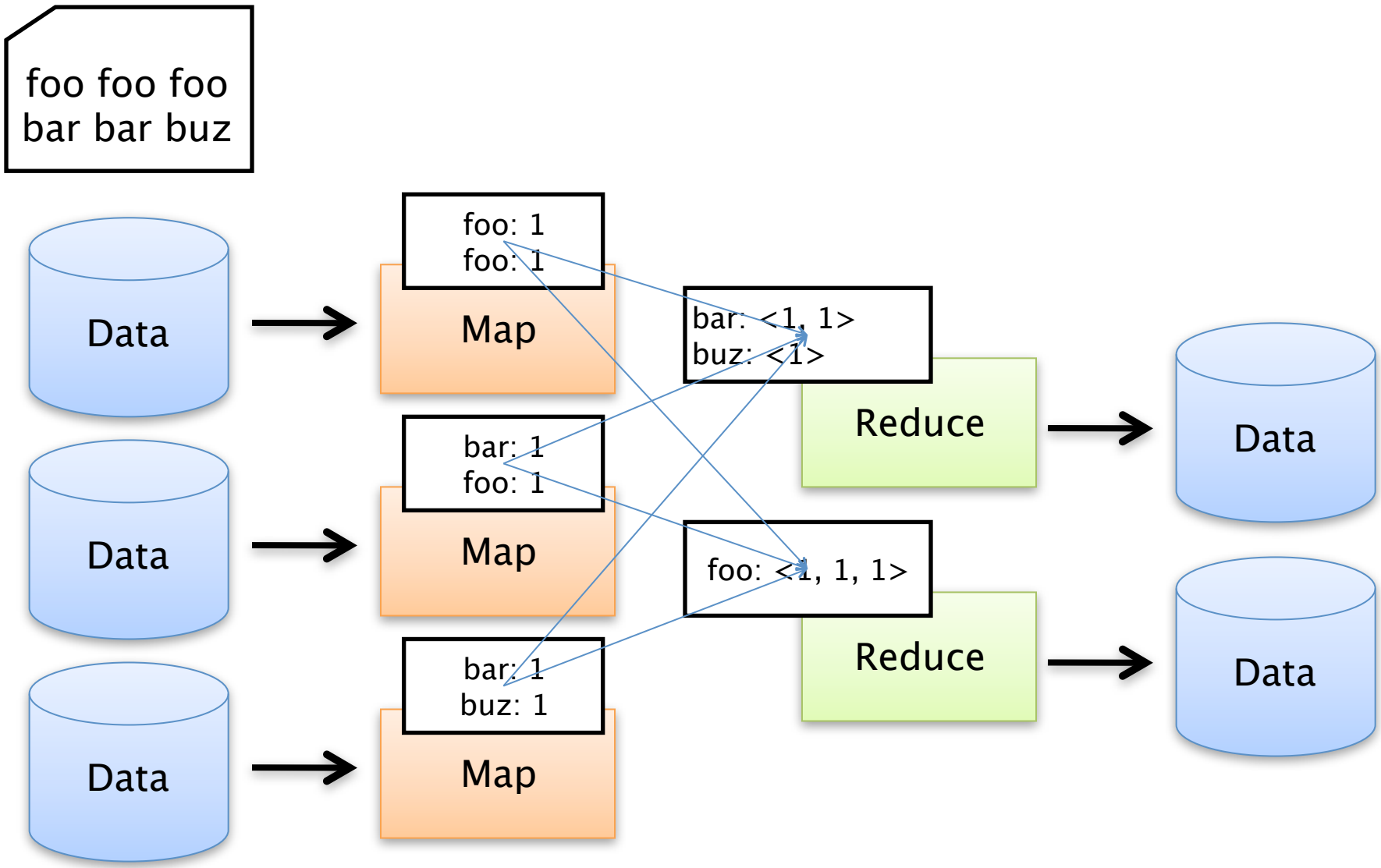
入力文書: doc1

foo foo foo
bar bar buz



例: ワードカウント

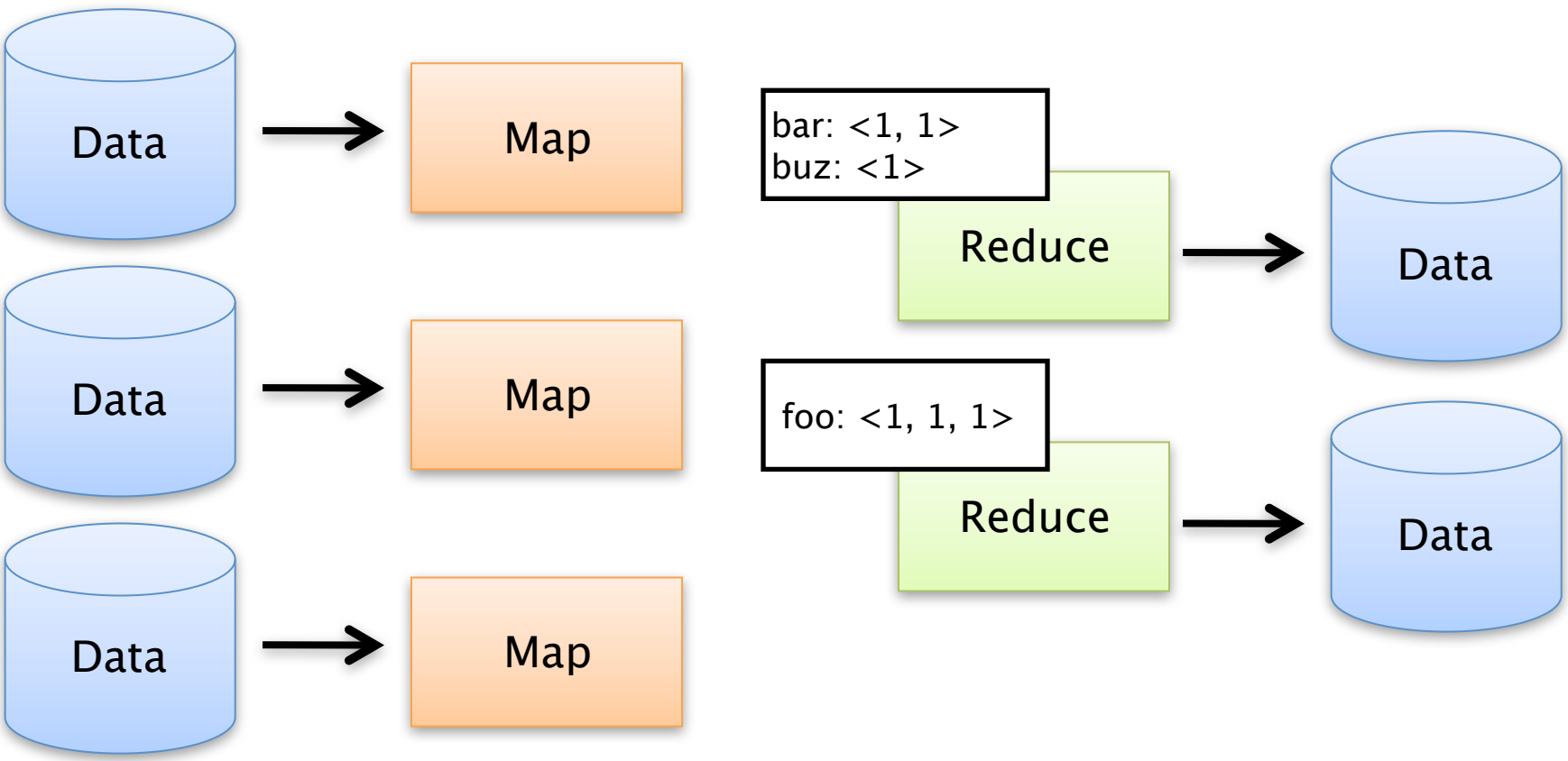
入力文書: doc1



例: ワードカウント

入力文書: doc1

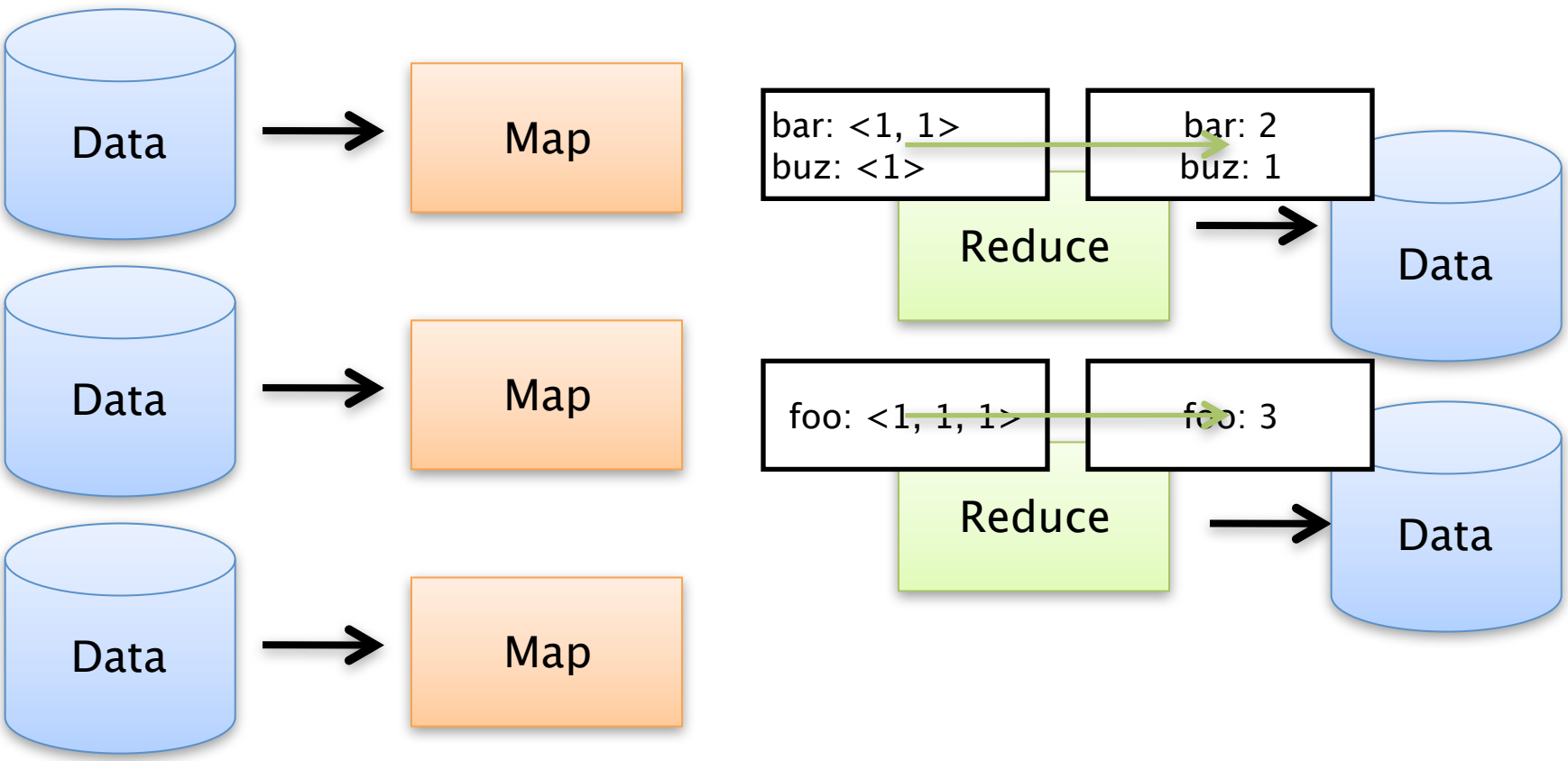
foo foo foo
bar bar buz



例: ワードカウント

入力文書: doc1

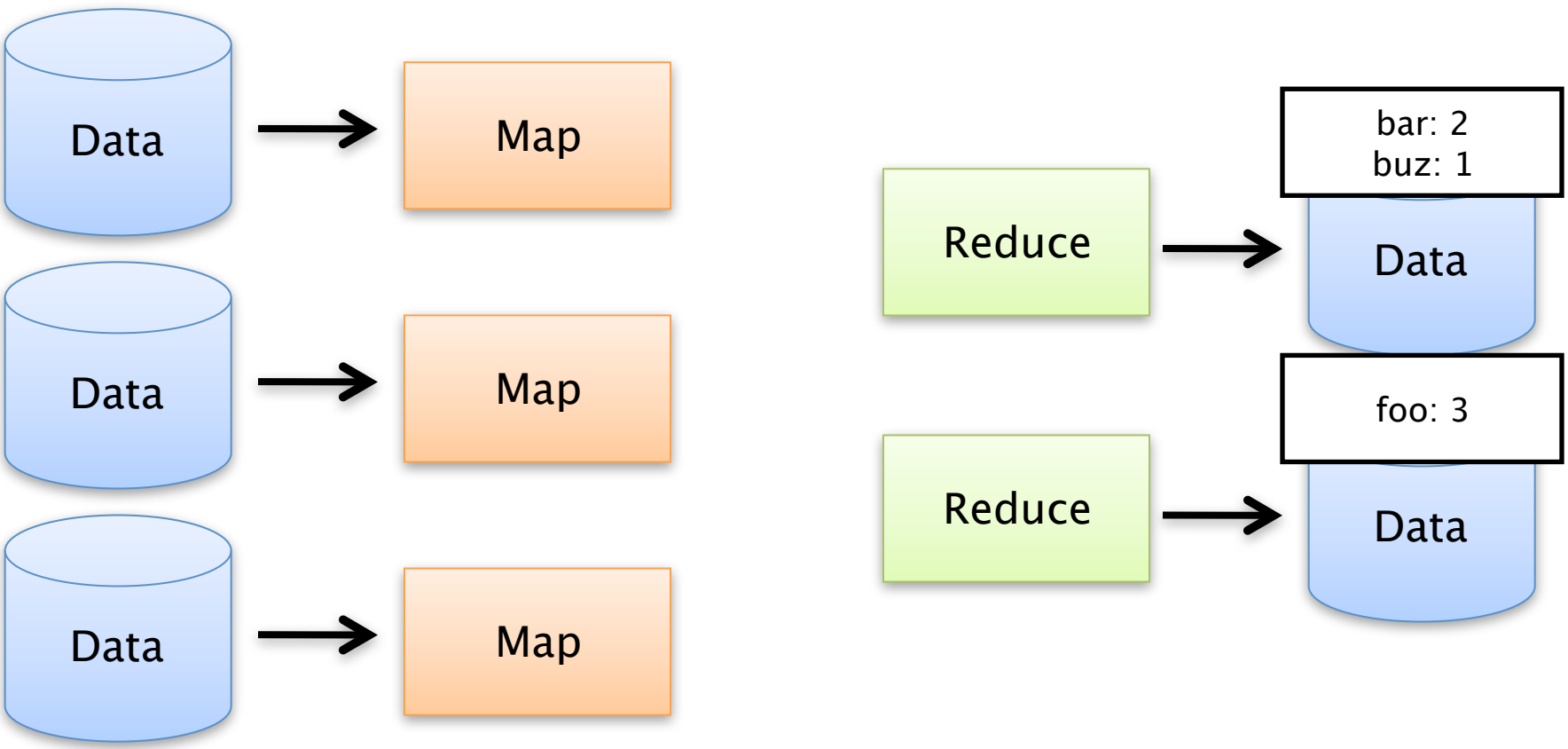
foo foo foo
bar bar buz



例: ワードカウント

入力文書: doc1

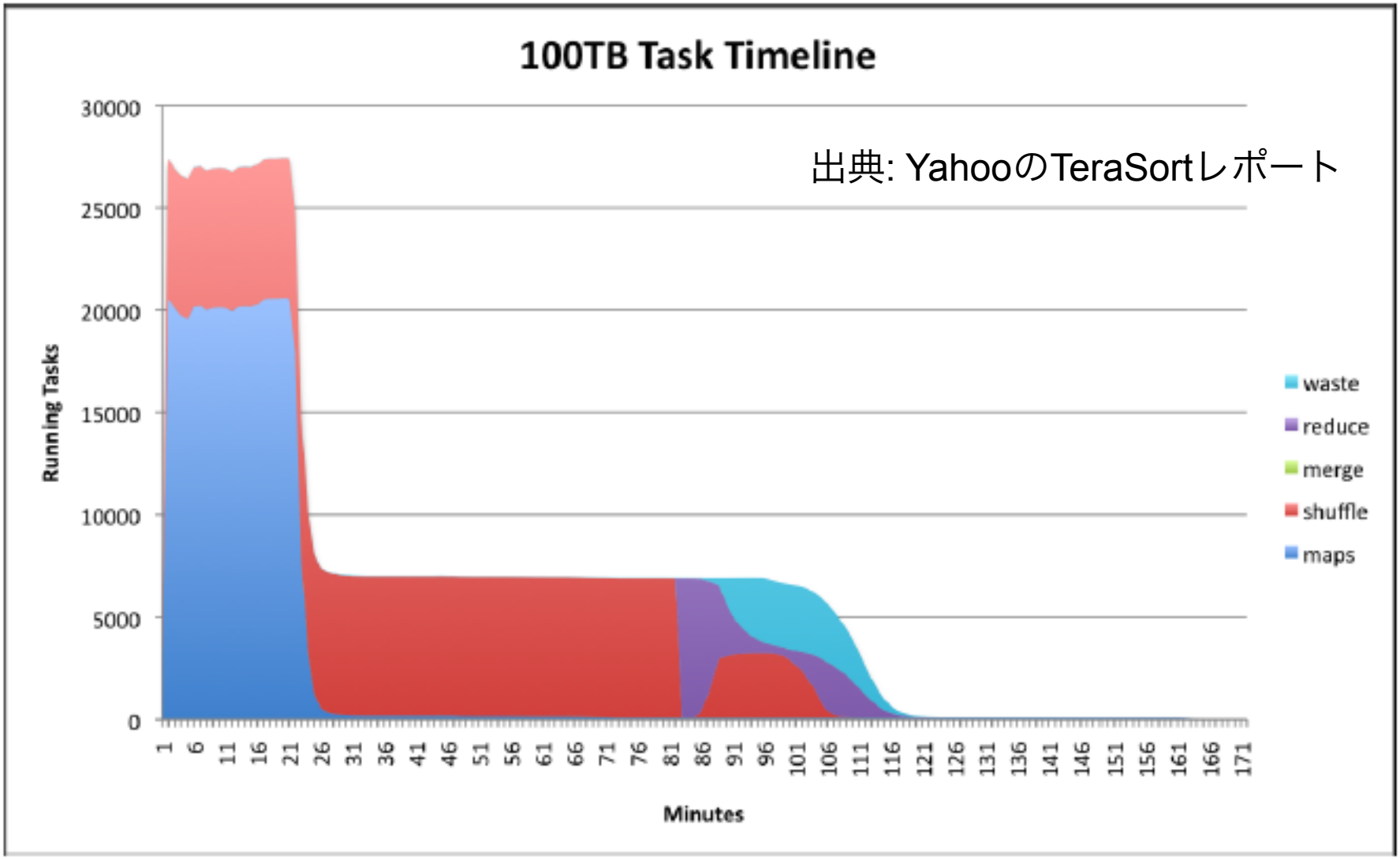
foo foo foo
bar bar buz



MapReduceの特徴

- データ通信
  - 各Map処理、Reduce処理は完全に並列に実行可能
  - マシンを増やせばその分処理能力が増える
- 耐故障性
  - 失敗したMap, Reduce処理は他のノードで再実行される
  - 遅いMap, Reduce処理についても同じ
- ローカリティ
  - データのある場所で計算を始めれば、ネットワークを使う必要がなくなる
    - Moving Computation is Cheaper Than Moving Data

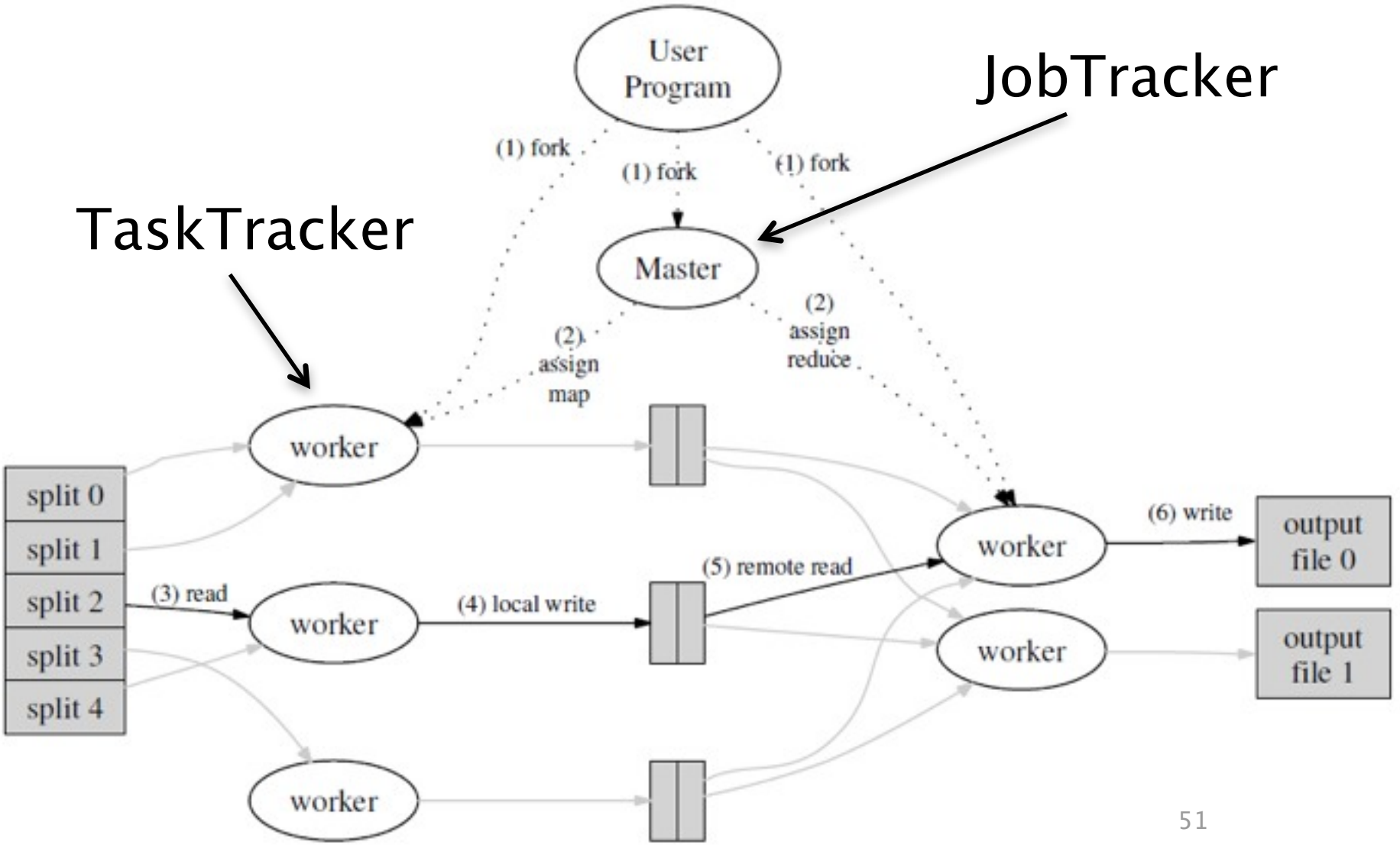
100TBソートの実行時間内訳



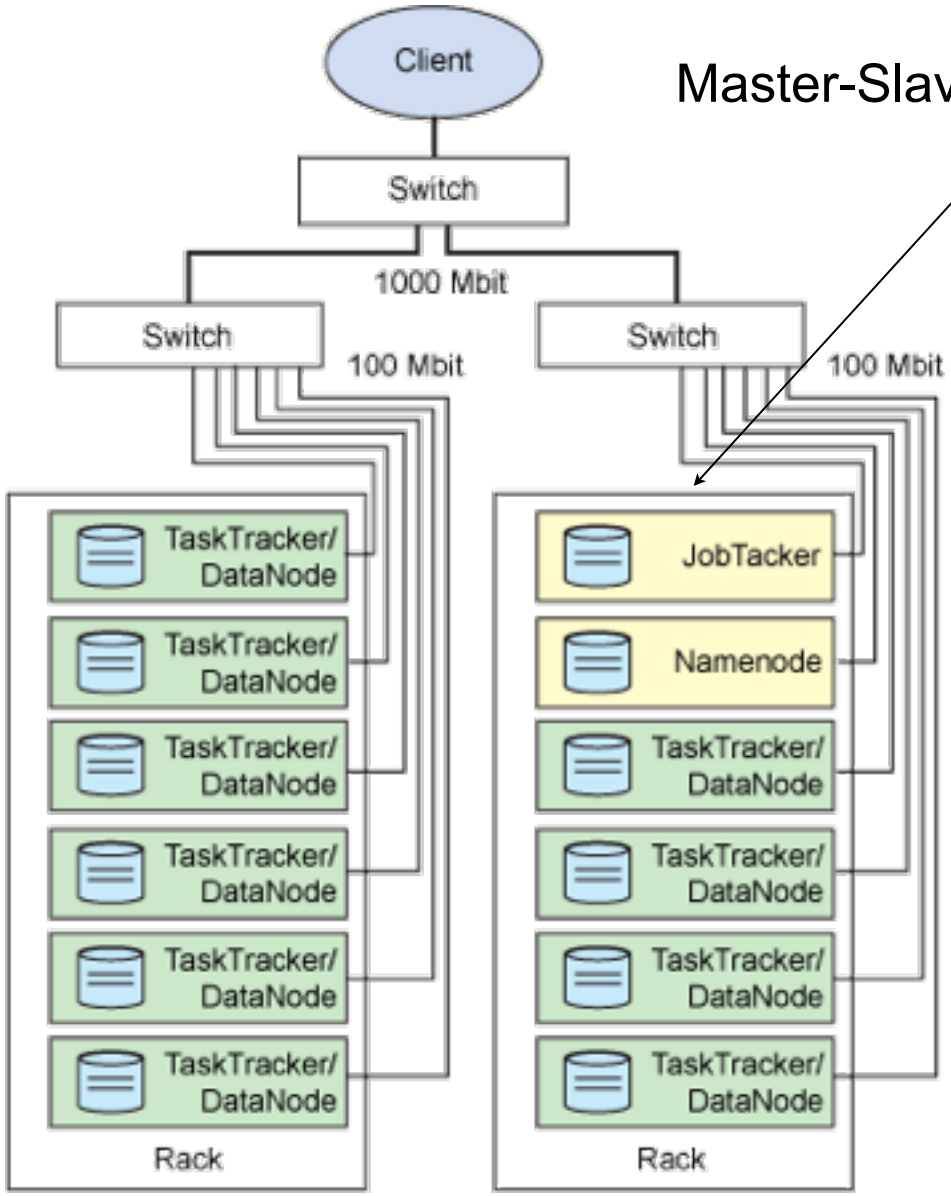
Hadoop MapReduce

- 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



Hadoopのシステム構成



Master-Slaveアーキテクチャ

MapReduce用の上位言語

Hive

- SQLライクな言語で、MapReduceジョブを記述
  - Javaを書かずに、必要なデータを得るためのジョブを簡単に作成できる

```
hive> CREATE TABLE shakespeare (freq INT, word STRING)
   ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED
   AS TEXTFILE;
hive> LOAD DATA INPATH "shakespeare_freq"
       INTO TABLE shakespeare;
hive> SELECT * FROM shakespeare LIMIT 10;
hive> SELECT * FROM shakespeare
  WHERE freq > 100 SORT BY freq ASC
  LIMIT 10;
```

Pig

- MapReduce用のDSL (Domain Specific Language)

Pig Latin

```
A = LOAD 'myfile'
    AS (x, y, z);
B = FILTER A by x> 0;
C = GROUP B BY x;
D = FOREACH A GENERATE
x, COUNT(B);
STORE D INTO 'output';
```

pig.jar:
- parses
- checks
- optimizes
- plans execution
- submits jar to Hadoop
- monitors job progress

Execution Plan
Map:
Filter

Reduce:
Count

Enjoy Hadoop ☺

Thank you!

第二部

Hadoopと既存システムとの連携

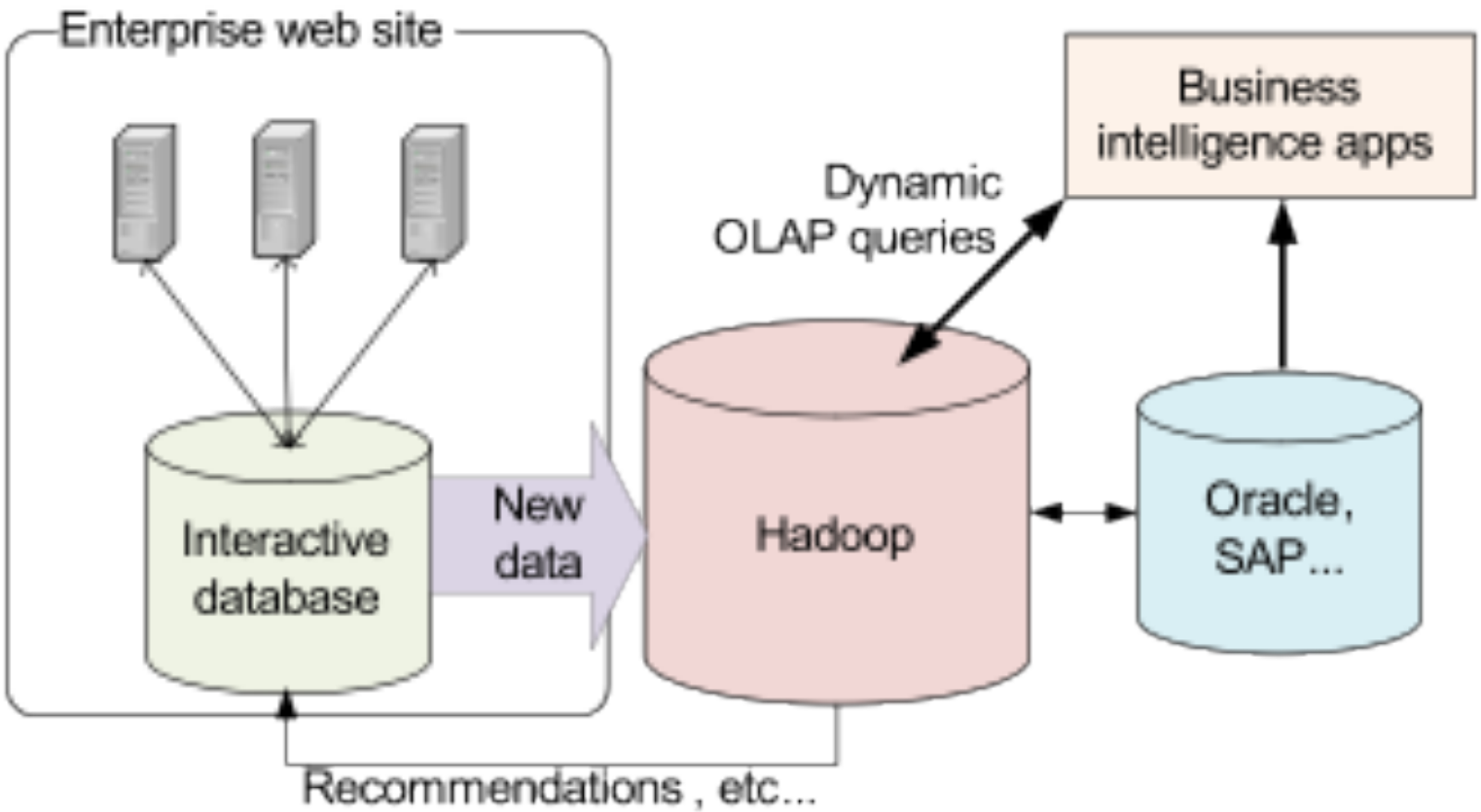
既存システムとHadoopの住み分け

- データセンターでは既に様々なコンポーネントが存在している
  - データベース
  - データウェアハウス
  - ファイルサーバー
  - バックアップシステム
- Hadoopはそのようなコンポーネントの1つでしかない
  - その中にhadoopをどのようにfitさせれば良いか？

RDBMS vs Hadoop

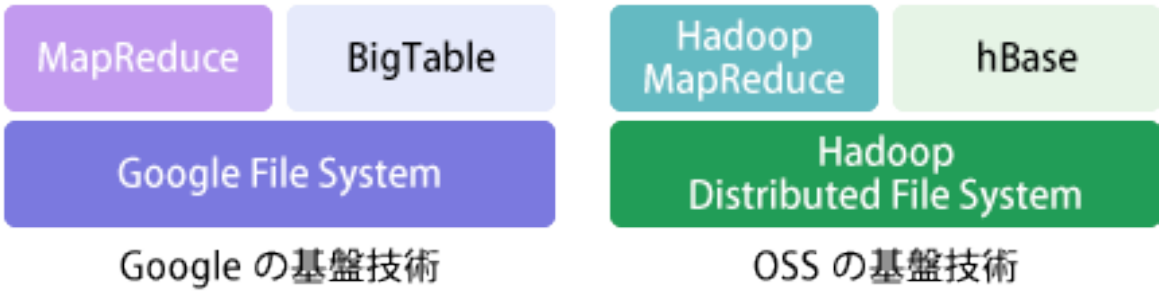
	RDBMS	Hadoop
Transactions/ second	1000's	n/a
Concurrent Queries	100's	10's
Update Patterns	Read / Write	Append Only
Join Complexity	100's of tables	Arbitrary keys
Schema Complexity	Structured	Structured or Unstructured
Total Data Volume	100's of TBs	10's of PBs
Per Job Data Volume	10's of TBs	10's of PBs
Processing Freedom	SQL	MapReduce, Streaming, Pig, Hive, etc.
Hardware Profile	High-end servers	Commodity / Utility Hardware

Hadoopの適用例


Enterprise web site
Business intelligence apps
Dynamic OLAP queries
Interactive database
New data
Hadoop
Oracle, SAP...
Recommendations , etc...

Hadoopのアーキテクチャ

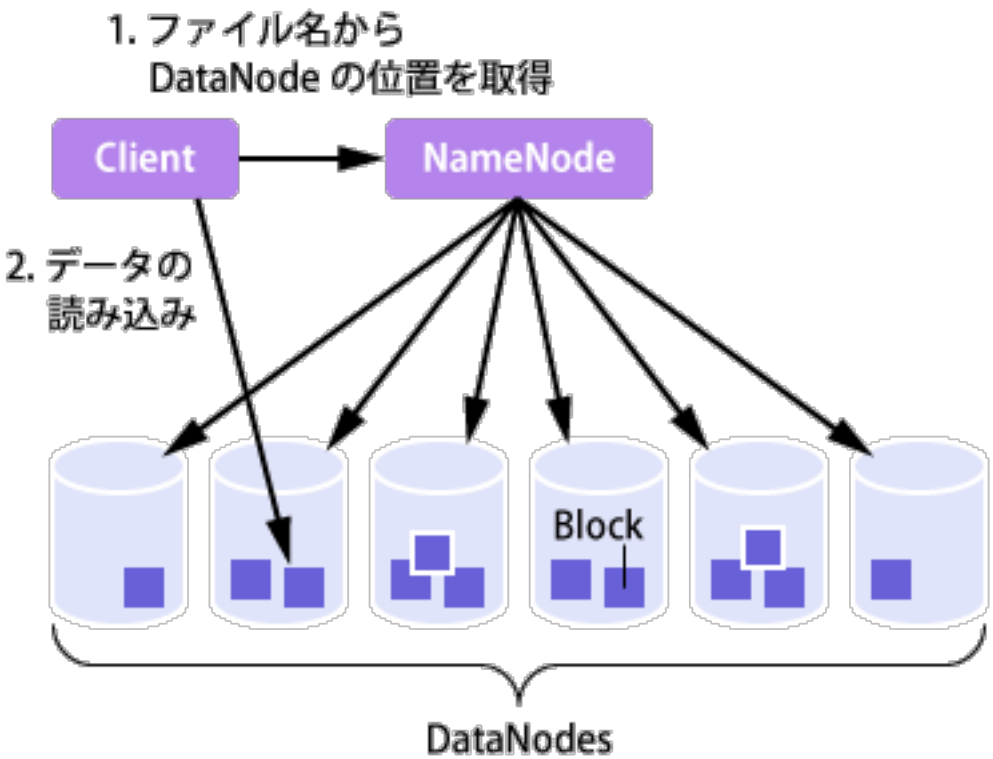
Hadoopの内部構成

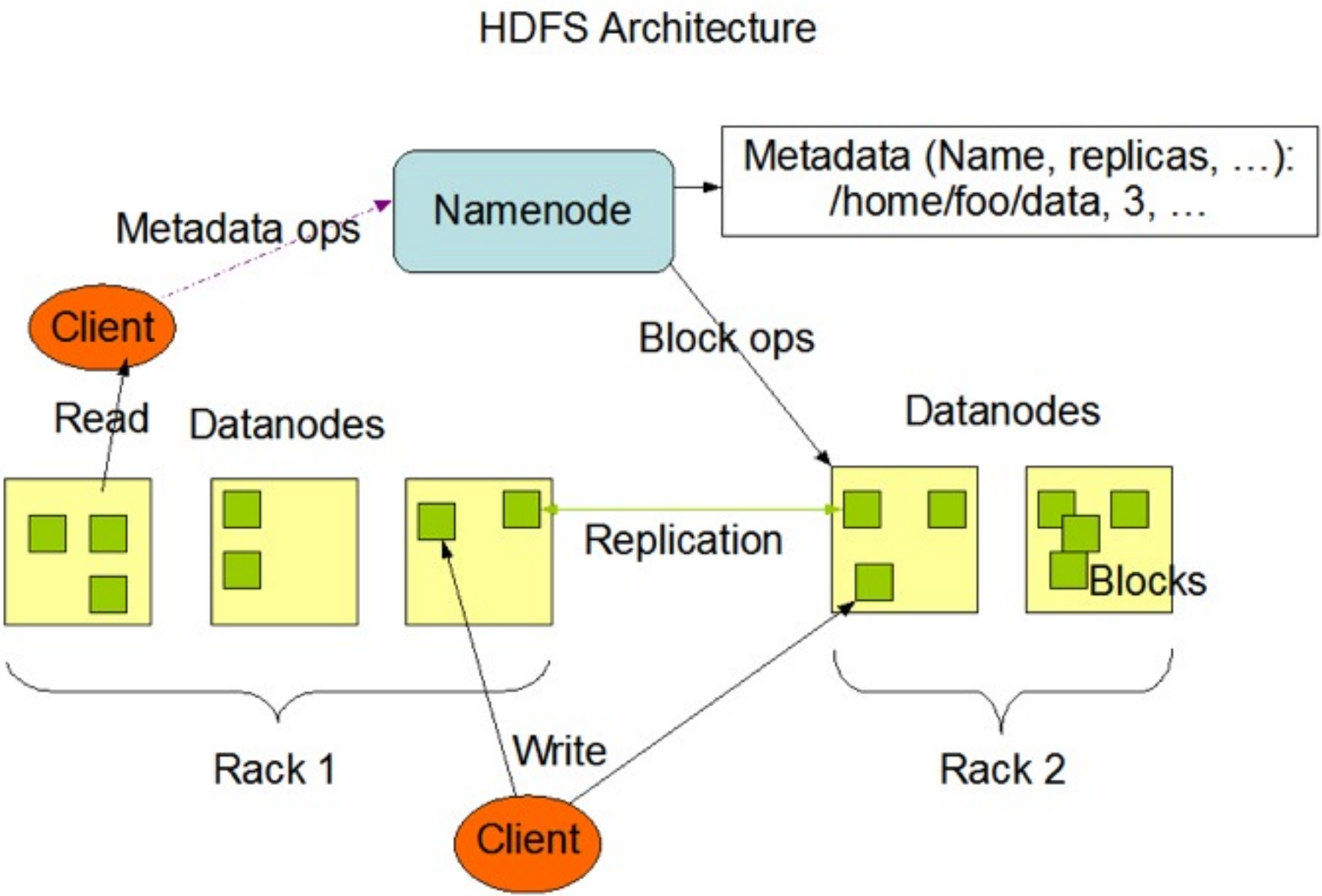


- Hadoop Distributed File System (HDFS)
  - GFSのクローン
  - MapReduceプログラムの入力や出力に使用
- Hadoop MapReduce
  - MapReduce実現するためのサーバー,ライブラリ

HDFS

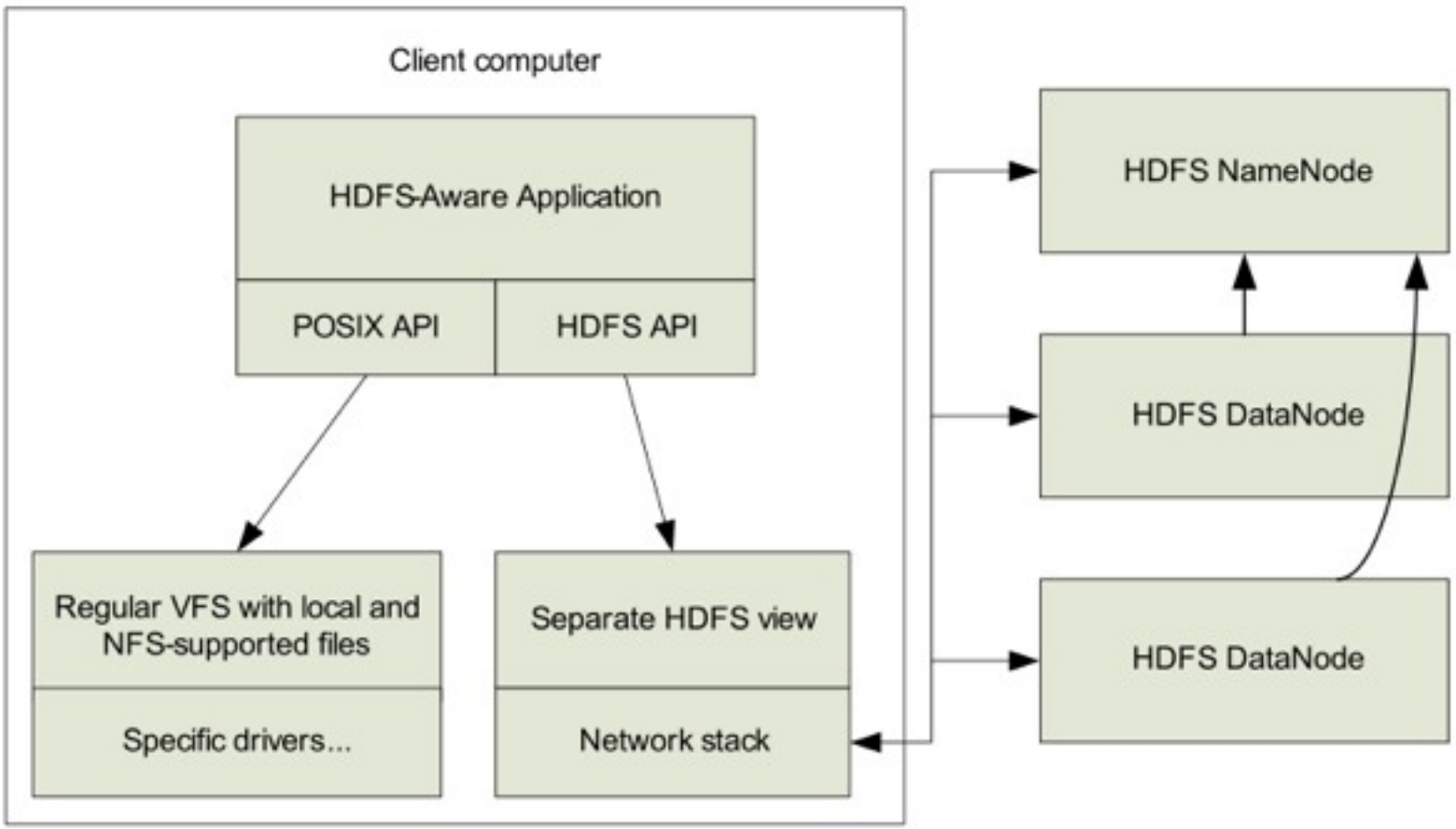
- Master/Slave アーキテクチャ
  - Masterが落ちるとシステム全体が停止
  - ファイルはブロック単位に分割して保存
    - 高スループット向き、低レイテンシ操作は苦手
- NameNode
  - Master
  - ファイルのメタデータ(パス・権限など)を管理
- DataNode
  - Slave
  - 実際のデータ(ブロックを管理)





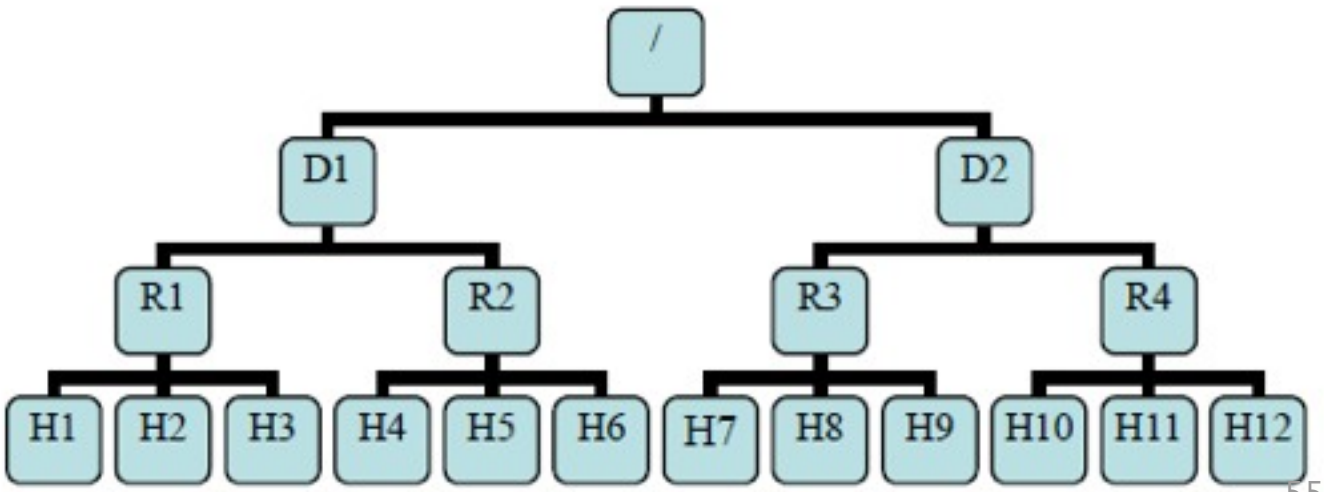
http://hadoop.apache.org/core/docs/current/hdfs_design.html

HDFS Clientのブロック図



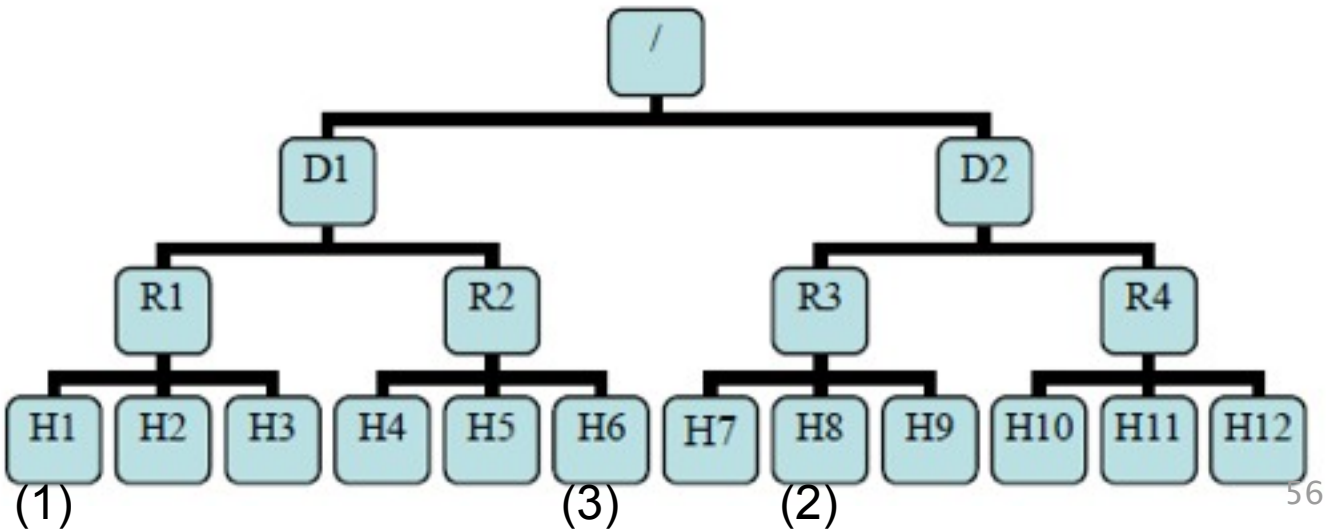
データ配置のアルゴリズム(1)

- あるデータをレプリケーション付きで書き込みたいとき、どのノードに配置するか？
  - 転送量を少なく
  - なるべく安全に (異なるラック・異なるDC)



データ配置のアルゴリズム(2)

- Hadoopが使用しているアルゴリズム
  - 1つ目は必ずローカルに書く
  - 2つ目は異なるDC(Rack)に書く
  - 3つ目は同じDC(Rack)の違うノードに書く
  - 4つ目移行はランダム



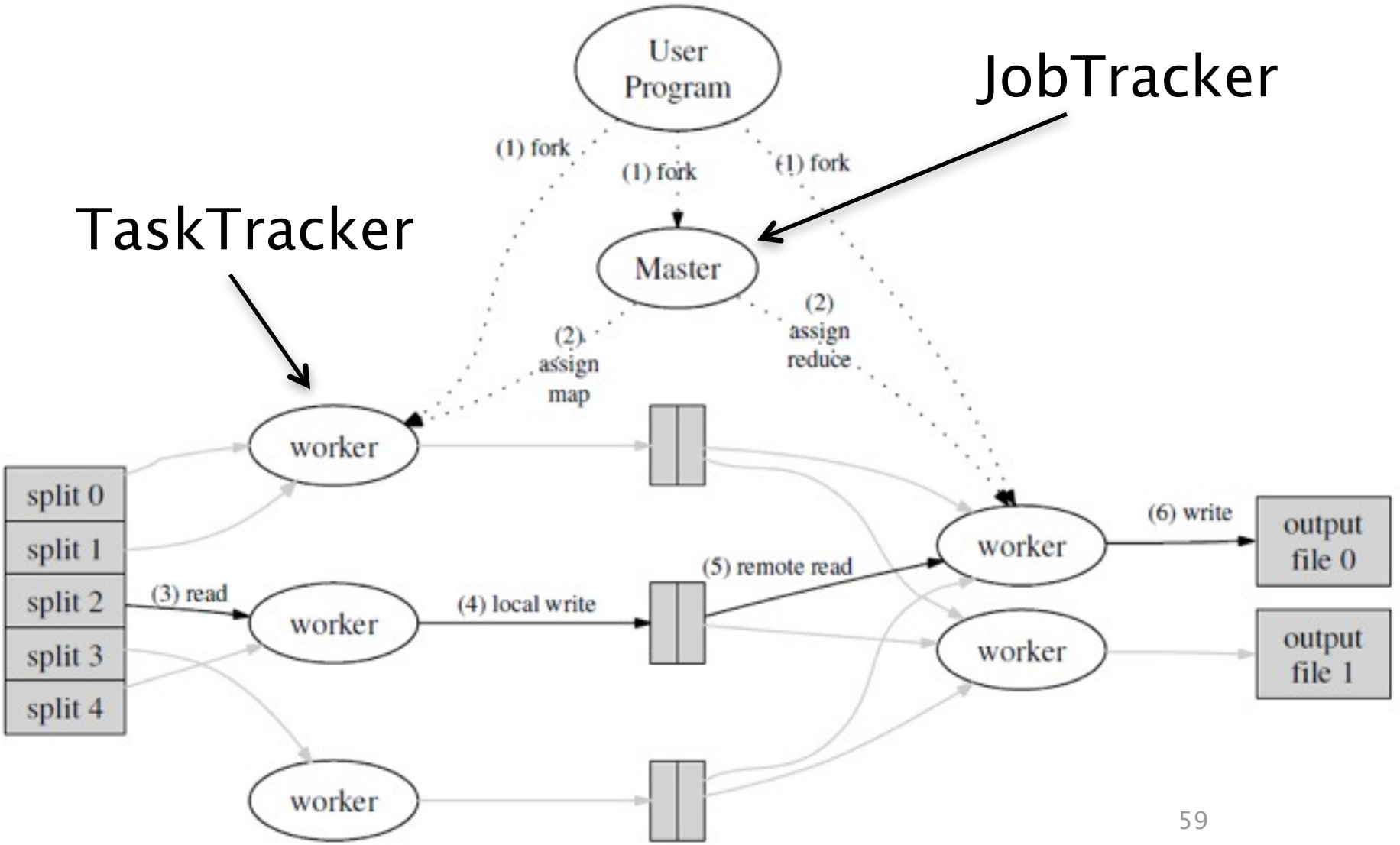
NameNodeの問題

- NameNodeがシステム全体のボトルネックになる
  - 大量のメタデータアクセスを裁く必要が有り、CPUパワーが必要
  - 大量のファイルを保存した場合、それに応じたメタデータを保持する必要が有り、メモリを大量に必要とする
  - 多数のデータノードからのハートビートメッセージを処理する必要が有り、大量のネットワーク接続を処理する必要が有る
  - Googleでも同様の問題を抱えている
    - http://queue.acm.org/detail.cfm?id=1594206
- 解決策
  - マルチマスター化? GoogleはBigTableにGoogleFileSystemのメタデータを置いているらしい?

Hadoop MapReduce

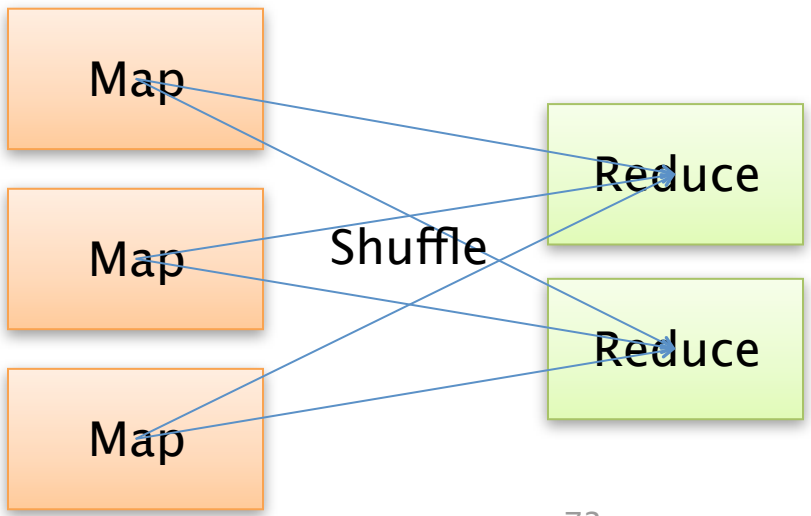
- Master/Slave アーキテクチャ
- JobTracker
  - Master
  - JobをTaskに分割し、Taskを各TaskTrackerに分配
    - Job: MapReduceプログラムの実行単位
    - Task: MapTask, ReduceTask
  - 全てのTaskの進行状況を監視し、死んだり遅れたりしたTaskは別のTaskTrackerで実行させる
- TaskTracker
  - Slave
  - JobTrackerにアサインされたTaskを実行
    - 実際の計算処理を行う

MapReduce Architecture



MapReduceの短所

- 処理は全てMapReduceの枠内に収める必要が有る
- Shuffleフェーズで大規模に通信が発生
  - 全Mapper <-> 全Reducerの通信
  - ネットワーク輻輳が起こり計算が進まなくなる
  - Shuffleフェーズで渡されるデータ量(Mapの出力)を削減するのが高速化へのポイント



Hadoopの周辺プロジェクト

Hive

- SQLライクな言語で、MapReduceジョブを記述
  - Javaを書かずに、必要なデータを得るためのジョブを簡単に作成できる

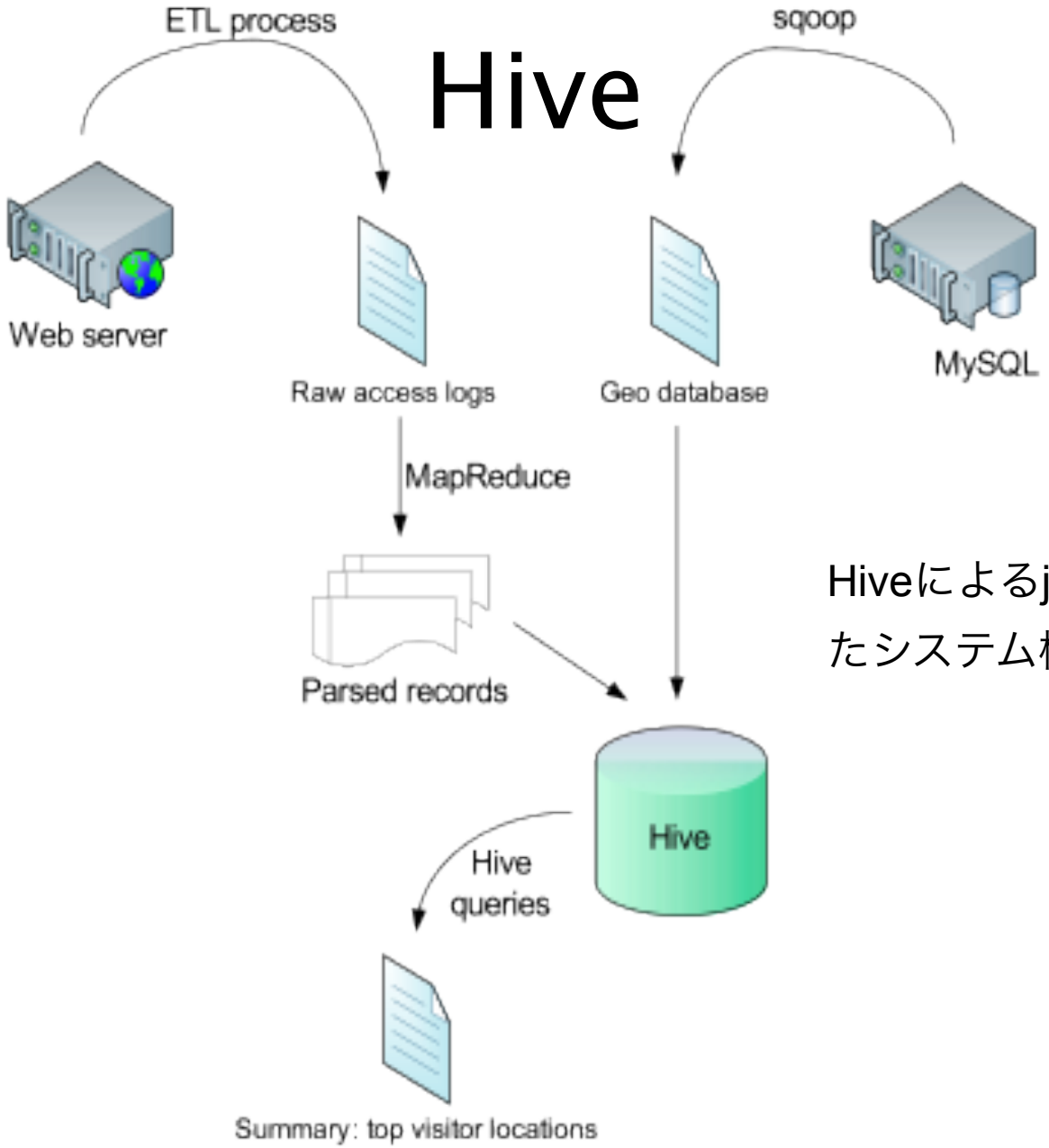
```
hive> CREATE TABLE shakespeare (freq INT, word STRING)
   ROW FORMAT DELIMITED FIELDS TERMINATED BY '\t' STORED
   AS TEXTFILE;
hive> LOAD DATA INPATH "shakespeare_freq"
       INTO TABLE shakespeare;
hive> SELECT * FROM shakespeare LIMIT 10;
hive> SELECT * FROM shakespeare
  WHERE freq > 100 SORT BY freq ASC
  LIMIT 10;
```

例: HiveによるJoin操作

- MapReduceを使用して、大規模なデータ同士のjoinを簡単に実行できる

```
hive> INSERT OVERWRITE TABLE merged
    SELECT s.word, s.freq, k.freq FROM
    shakespeare s JOIN shakespeare2 k ON
    (s.word = k.word)
    WHERE s.freq >= 1 AND k.freq >= 1;

hive> SELECT * FROM merged LIMIT 20;
```

Hive



HiveによるjoinをHiveを使用したシステム構築例

Pig

- MapReduce用のDSL (Domain Specific Language)

Pig Latin

```
A = LOAD 'myfile'
    AS (x, y, z);
B = FILTER A by x> 0;
C = GROUP B BY x;
D = FOREACH A GENERATE
x, COUNT(B);
STORE D INTO 'output';
```

pig.jar:
- parses
- checks
- optimizes
- plans execution
- submits jar to Hadoop
- monitors job progress

Execution Plan
Map:
Filter

Reduce:
Count

例: Pigによるデータ操作

- Deta-flow指向言語
  - データ型としてset, associative array, tuple等をサポート
- スクリプト例:

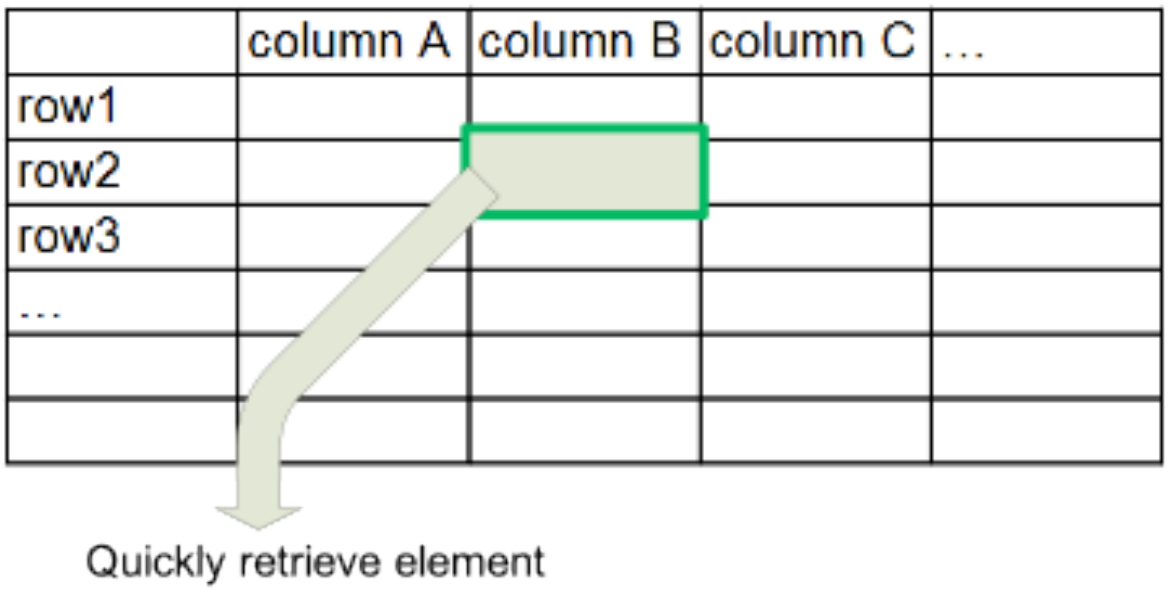
```
# 入力データの生成
named_events = FOREACH events_by_time GENERATE $1
   as event, $2 as hour, $3 as minute;
# 12時台のイベント

noon_events = FILTER named_events BY hour = '12';
# uniqueなイベント

distinct_events = DISTINCT noon_events;
```

hBase: 分散データベース

- 列指向データベース
  - Google BigTableのデザインを踏襲して実装
  - データに対するinteractiveなアクセスを提供
- 非常に大規模なデータを扱うのに適している
  - 数TB〜数PB
- 制約されたアクセスモデル
  - keyでのlookup
  - transactionは行単位となっており、通常のRDBMSと比べると非常に制限されている

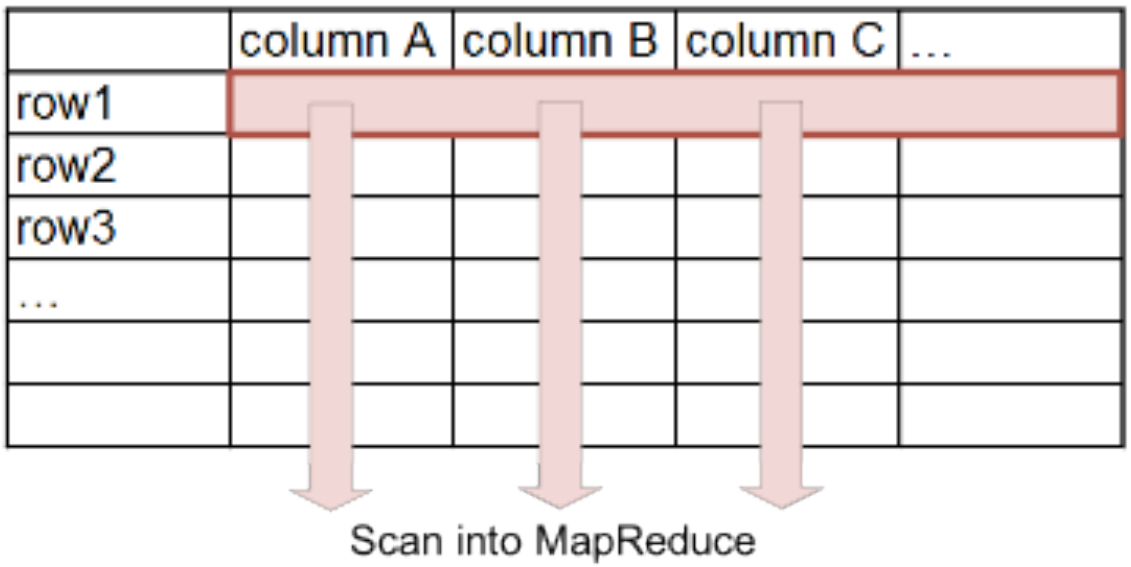
hBase: 単一行へのアクセス

- 単一行への、keyによるアクセスが非常に高速
  - 特にWebアプリケーションでは重要になるデータlookupの形式



hBase: MapReduceの入力

- 各rowがMapReduceの入力となる
  - MapReduceジョブで、sort/search/indexing等を行うことが出来る。
- hBaseのsequentialなscanを得意としているため、MapReduceジョブの速度は低下しない
  - オンラインデータ処理と高速なバッチ処理の両方が実現可能になる。



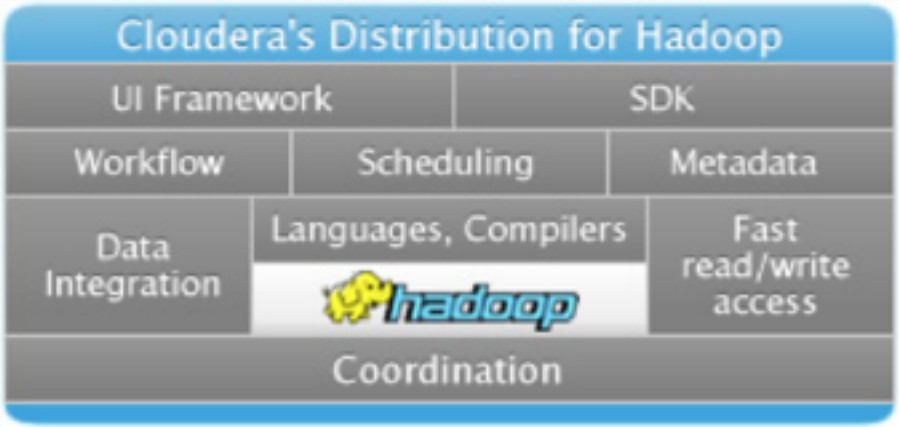
More and more projects...

- fuse-HDFS: HDFSをfuseマウントするプログラム
- Zookeeper: 分散合意エンジン
- Sqoop: RDBMSからHDFSへの取り込みエンジン
- Avro: Serialization + RPCフレームワーク
- Scribe, Flume: ログ収集フレームワーク
- Mahout: MapReduceを使用した機械学習ライブラリ
- Oozie: ワークフローエンジン
- ...
- 全て現実問題を解決するために、企業手動で開発

Hadoopの使用方法

インストール (CDH)

- Cloudera Distribution for Hadoopが便利
  - Apache Projectで配布されているものに、Cloudera社が独自にbugfix & security fixを施したパッケージ (無料)
  - yum/apt等のパッケージシステム経由でのインストール
    - yum install hadoop-0.20-*
  - Hive/Pig/Hbase/Flume等の周辺ソフトも含まれている



HDFSの操作方法

```
# ls
alias dfsls='~/hadoop/bin/hadoop dfs -ls‘

# ls -r
alias dfslsr='~/hadoop/bin/hadoop dfs -lsr‘

# rm
alias dfsrm='~/hadoop/bin/hadoop dfs -rm‘

# rm -r
alias dfsrmr='~/hadoop/bin/hadoop dfs -rmr'

# cat
alias dfscat='~/hadoop/bin/hadoop dfs -cat‘

# mkdir
alias dfsmkdir='~/hadoop/bin/hadoop dfs -mkdir‘
```

HDFSの操作方法

- HDFS上にファイルを転送

```
alias dfsput='~/hadoop/bin/hadoop dfs -put‘
dfsput   <local-path>  <hdfs-path>
```

- HDFS上からファイルを転送

```
alias dfsget='~/hadoop/bin/hadoop dfs -get‘
dfsget  <hdfs-path>  <local-path>
```

HadoopStreaming

- 標準入出力を介してMapReduce処理を書けるようにするための仕組み
  - sh ・ C++ ・ Ruby ・ Pythonなど、任意の言語でMapReduceが可能になる
  - http://hadoop.apache.org/core/docs/r0.15.3/streaming.html
- Hadoop Streamingは単純なwrapperライブラリ
  - 指定したプログラムの標準入力に<key, value>を渡す
  - 標準出力から結果を読み込み、それを出力
- Amazon, Facebook等でもStreamingをよく使用している
  - http://wiki.apache.org/hadoop/PoweredBy

使い方

- 実行方法

./bin/hadoop jar contrib/hadoop-0.20.2-streaming.jar

-input inputdir [HDFSのパス]

-output outputdir [HDFSのパス]

-mapper map [mapプログラムのパス]

-reduce reduce [reduceプログラムのパス]

-inputformat [TextInputFormat
| SequenceFileAsTextInputFormat]

-outputformat [TextOutputFormat]

例: Rubyによるワードカウント

```
$ ./bin/hadoop
   jar contrib/hadoop-0.20.2-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}“ }
```

MapReduceアルゴリズム
(1) Join操作

Join操作とは?

- ２つのデータセットが有ったときに、片方のデータがもう片方のデータを参照している。このとき、参照ではなくデータ自体で情報を結合したい。
- 例:
  - 入力:
    - EMP: 42, 太田, loc(13)
    - LOC: 13, 本郷三丁目
  - 出力
    - EMP: 42, 太田, loc(13), 本郷三丁目

MapReduceによるjoin

- Map-Side Join
  - 片方の表がメモリに載る範囲で有る場合、mapperに全てのデータを持たせてそこでjoinを行う
  - 両方のテーブルが大量のデータの場合に対応出来無い
- Reduce-Side Join
  - reducer側でjoinを行う
  - 次スライド以降で説明

```
String getLocation(int locId) {
   // メモリ上の構造 or 外部データベースを参照
}
void map(k, v) {
   int locId = parse_locid(v);
   String location = getLocation(locId);
}
```

map-side join

Reduce-Side Join: データ構造

- Union構造

```
class Record {
    enum Type { emp, loc }
    Type type;

    // EMP用メンバ

    String empName;
    int empId;

    // LOC用メンバ

    String locationName;
    int locId;
};
```

Reduce-Side Join: Mapper

- 各テーブル毎にmapperを走らせ、同じreducerを使用する。mapperではデータtypeを設定する。keyはlocIdを指定。

```
void map(k, v) { // employee用
    Record r = parse(v);
    r.type = Type.emp;
    emit (r.locId, r);
}
void map(k, v) { // location用
    Record r = parse(v);
    r.type = Type.loc;
    emit (r.locId, r);
}
```

Reduce-Side Join: Reducer

- valuesのなかで、Type.locのモノに場所情報が入っているので、それをemployeesに付与する。

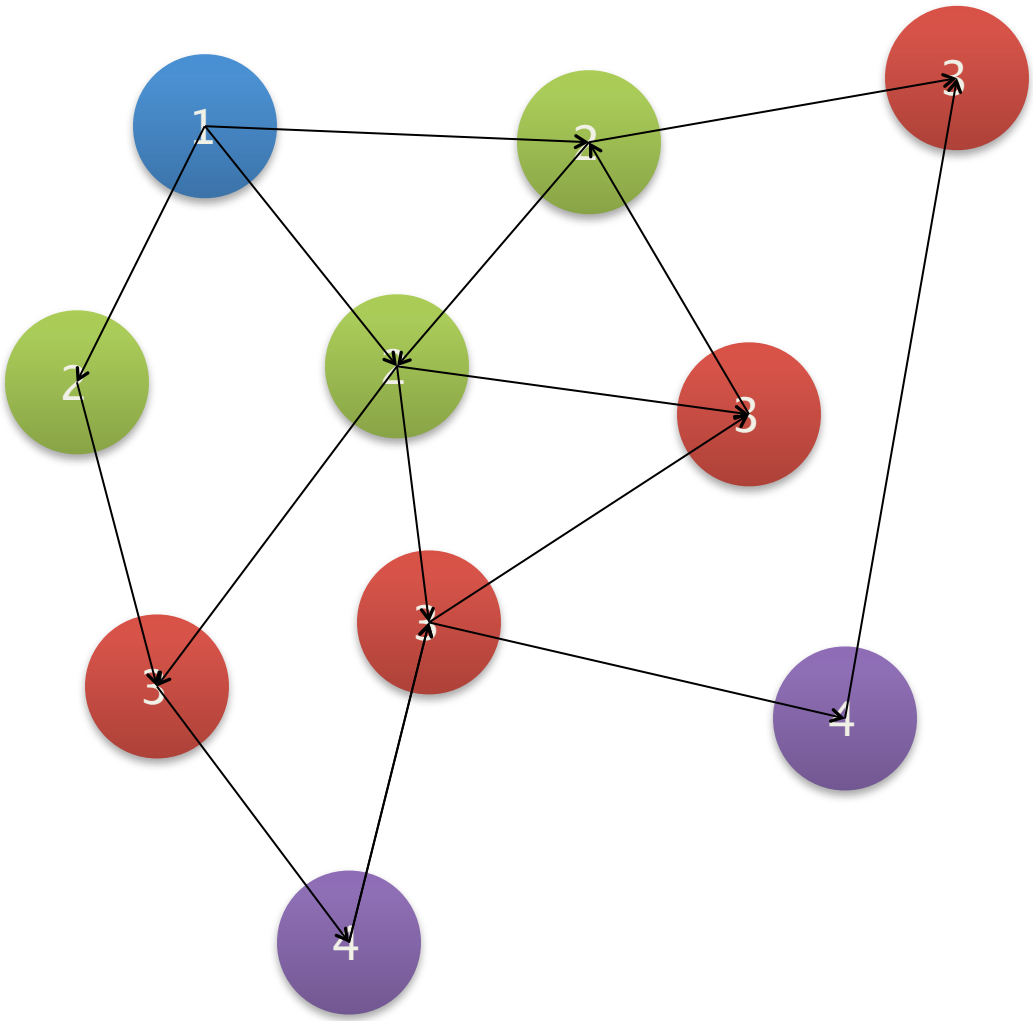
```
void reduce(k, values) {
  Record thisLocation; List<Record> employees;
  for (Record v in values) {
    if (v.type == Typ.loc)
      thisLocation = v;
    else
      employees.add(v);
  }
  for (Record e in employees) {
    e.locationName = thisLocation.locationName;
    emit(e);
  }
}
```

MapReduceアルゴリズム
(2) グラフアルゴリズム

例: MapReduceでの並列BFS

- グラフ上の1頂点から、1つ以上の頂点集合までの最短パスを求める
- アルゴリズム
  - DistanceTo(StartNode) == 0
  - DistanceTo(n) == 1
    - ただしnはStartNodeから辿れる
  - DistanceTo(m) = 1 + min(DistanceTo(n), n ∈ S)
    - 頂点集合Sから辿れる全ての頂点m
- MapReduce型に落とすにはどのようにすればいいか？
  - 行列の保持方法
  - Mapper, Reducerではどのような処理が必要か?

ノード1からの距離を可視化



隣接リストでデータを保持

隣接行列を構築し、ゼロを取り除く

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

1: 2, 4
2: 1, 3, 4
3: 1
4: 1, 3

Parallel BFS by MapReduce

- Map
  - 入力
    - Key: 頂点n
    - Value: D (startからの距離), S (nから到達可能なノード一覧)
  - 出力
    - Sに含まれる全てのノードmについて(m, D + 1)を出力
- Reduce
  - 入力: ノードmにへの経路列
  - 出力: 経路の中で最短のものを出力する
- MapReduce1回で、1 hop進める事ができる。出力を同じMapReduceプログラムの入力として再度使用し、収束するまでジョブを走らせ続ける。

まとめ

- Hadoopはグーグルの基盤技術のOSSクローン
  - 様々な企業/団体によって開発・改良・利用されており、新しい周辺プロジェクトがどんどん誕生している
- MapReduceは大量データ処理に適したモデル
  - ジョブの失敗や並列化等を自動的に行ってくれる
  - 全てではないが、非常にフィットするアルゴリズムが有る
  - 例としてJOIN操作やグラフ上の探索問題等を解くことが出来る
  - 単一マスター, Shuffle時のネットワーク転送問題
- 是非、余っているマシン等で試してみて下さい:-)

Enjoy Playing Around Hadoop ☺

Thank you!