

超入門 大規模分散処理フレームワーク Hadoop

SRA OSS, Inc. 日本支社
技術開発部 エンジニア
長田 悠吾

Cloudera Certified Developer for Apache Hadoop
Cloudera Certified Administrator for Apache Hadoop

Hadoopとは

- 大規模なデータを並列分散処理を行うフレームワークを提供
- Googleによる MapReduce および Google File System(GFS) の論文をベースに開発されたApacheプロジェクトのOSS

Google



MapReduce

MapReduce

分散処理フレームワーク

GFS

HDFS

分散ファイルシステム

Hadoopの歴史

- 2003年 Google GFS の論文発表
- 2004年 Google MapReduce の論文発表
- 2005年 Hadoop プロトタイプの誕生
- 2006年 20ノードで動作
- 2006年 Yahoo!が本格的に注力
- 2007年 200ノードで稼働
- 2008年 Apacheトップレベルプロジェクトへ昇格
- 2008年 大規模なソート処理で世界記録樹立

910台のノードで 1TBデータをソート 297秒 → 209秒!

- 2008年11月 Google 1TBデータをソート 68秒
- 2009年 1月 Hadoop 1TBデータをソート 62秒
- 現在 Yahoo! 4000台、Facebook 2250台以上

大きな進歩
企業によって
育てられたOSS

Hadoop が注目された背景

大規模データの保管と大規模データの分析

- 時間の経過につれてデータ量が爆発
 - バックアップできない!

データの冗長性

- 膨大な情報から価値ある情報を抽出するニーズ
 - Webショッピングのリコメンダ
 - アクセスログ解析での行動分析

大規模データの各データ間の分析 マーケティングデータとして活用

ビッグデータ！

- ニューヨーク株式市場
 - 毎日 1TB の取引データ
- Facebook
 - 100億枚の写真 = 1PB の記憶容量
- 欧州原子核研究機構（CERN）
 - 毎日 40TB, 年間で 15 PBのデータ
- オンラインショッピング、ブログ、twitter, ...

大規模データの処理

- ディスク容量の向上、CPU パワーの向上
 - >> ディスクアクセス性能の向上
 - データを処理しきれない
 - CPUを使い切れない
- データを分散
 - ⇒ ディスク I/O を分散、CPUの有効利用
 - 障害発生への対応が必要
 - データの同期、統合が困難

RDBMSの得意不得意

- 得意

- 細かいデータのインデックス高速検索
- 細かいデータの更新が得意
- データ一貫性の保証

- 不得意

- 大規模処理 ディスクI/O以上の性能は出せない
- スケールアウト[並列分散処理]が困難
- データ可用性を高めることが困難
- クラスタリングの運用が困難

可用性・性能を求めると
コストが増大

Hadoopの得意不得意

- 得意
 - 高スループット処理
 - 高スケーラビリティ
 - 高可用性
 - 安価なサーバで構成可能〔一般に調達できるサーバ〕
- 不得意
 - RDBMSが得意なトランザクション一貫性保証
 - データの更新(出来ない)
 - OLTP処理のような”秒”以下のレイテンシが必要な処理
(Hadoopの処理は”分”のオーダーで処理が終わる)

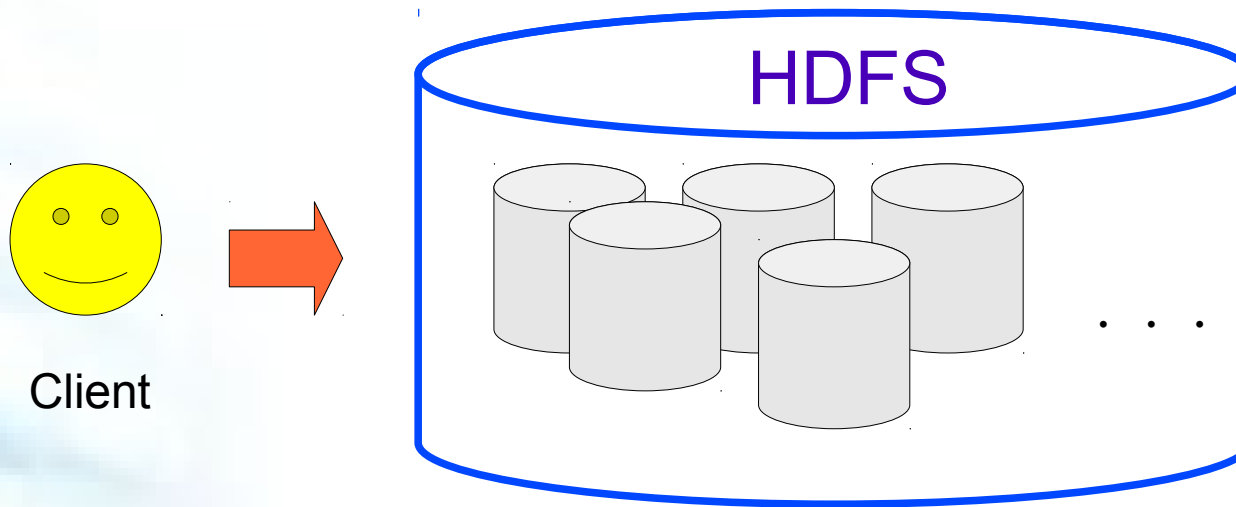
Hadoop の2大構成要素

- HDFS
 - Hadoop のファイルシステム
 - 大規模分散ストレージ
- MapReduce
 - Hadoop の分散処理フレームワーク

Hadoop のファイルシステム

HDFS: Hadoop Distributed File System

- 大規模分散ストレージ
 - 大規模データを複数のサーバに分散して保存
 - クライアントからは1つのストレージとして見える
 - サーバを増やすことで拡張可能 (= スケールアウト)

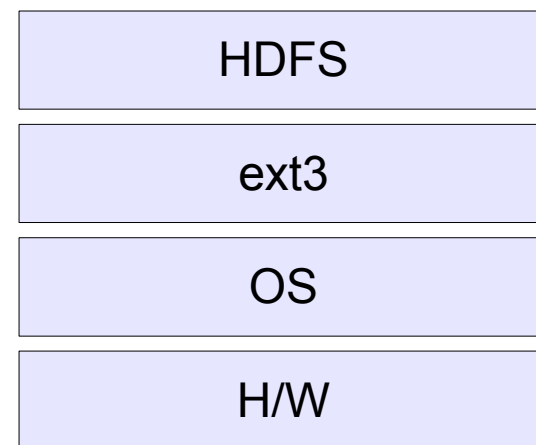


Hadoop のファイルシステム

HDFS: Hadoop Distributed File System

hadoop コマンド

- 通常のファイルシステム上で動作
 - 各サーバに保存されているデータは通常のファイル



- 普通のファイルシステムと似たコマンドで扱える
 - `hadoop fs` コマンド

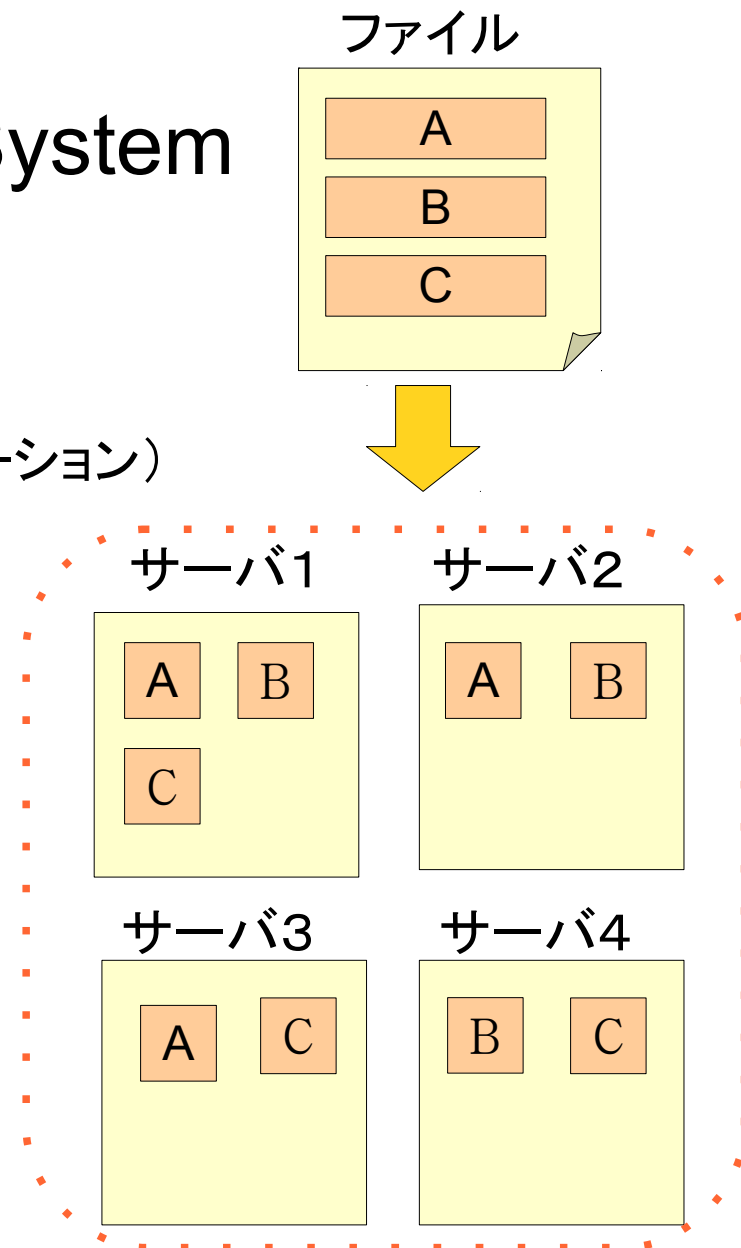
```
$ hadoop fs -put local_file.txt hdfs_file.txt
$ hadoop fs -ls
$ hadoop fs -mkdir mydata
$ hadoop fs -cat hdfs_file.txt
```

Hadoop のファイルシステム

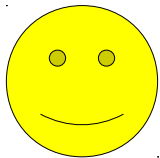
HDFS: Hadoop Distributed File System

- 冗長性
 - 1つのファイルを複数のブロックに分割して
同じブロックを複数のサーバーに複製(レプリケーション)
 - デフォルトのレプリケーション数は 3

- アクセスパターンの制限
 - Write-Once
 - ファイルの更新は行わない
 - シーケンシャルな読み込みを前提
 - ブロックサイズ 64MB
 - ランダムアクセスを想定していない

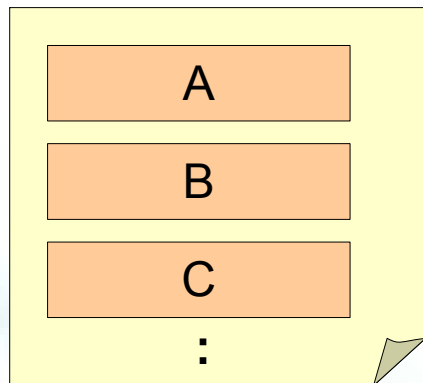


HDFS クライアント HDFS

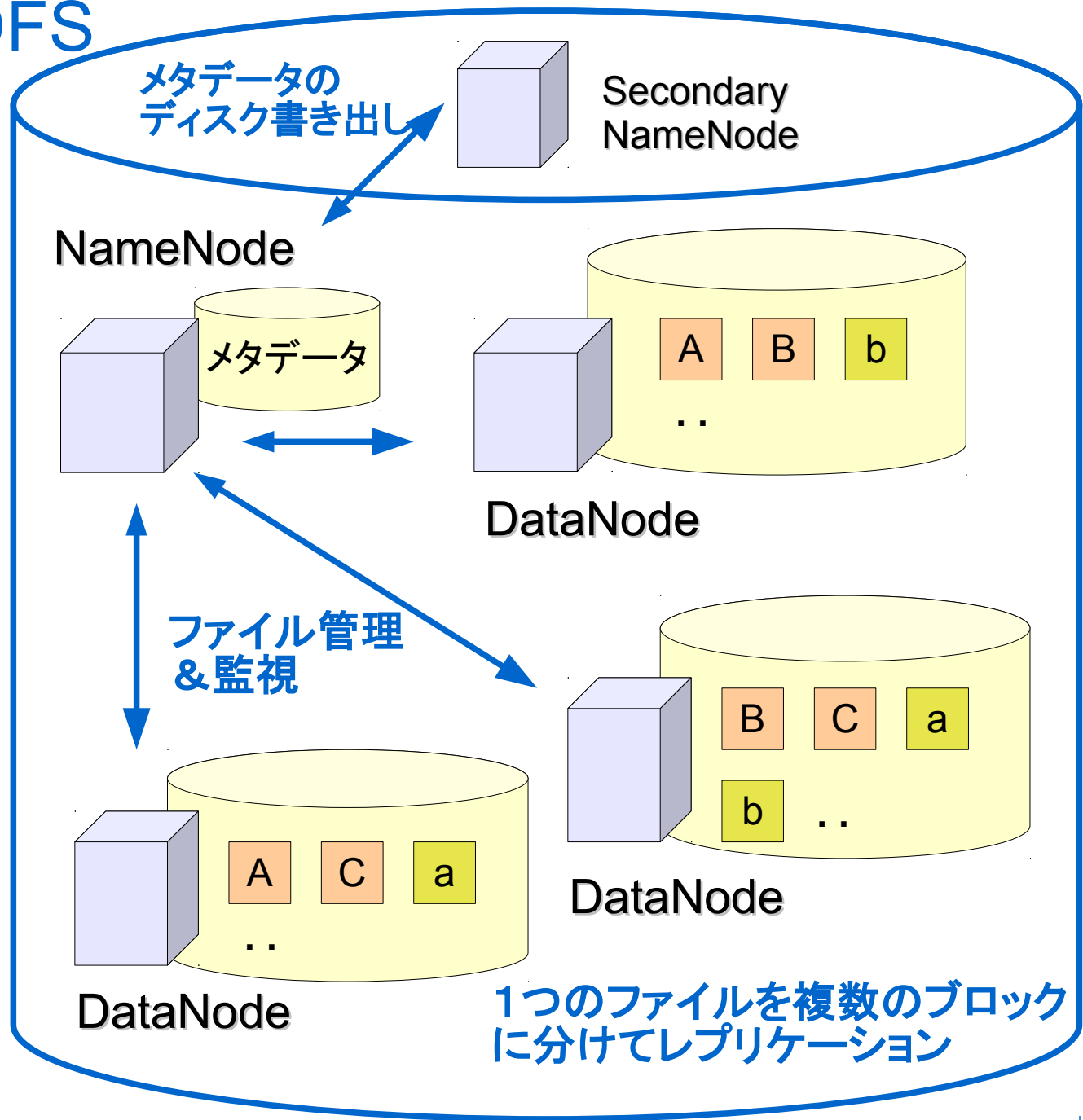
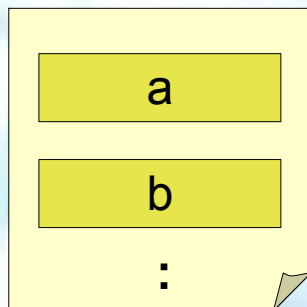


単一ストレージのように
ファイルを読み書き可能

ファイル1

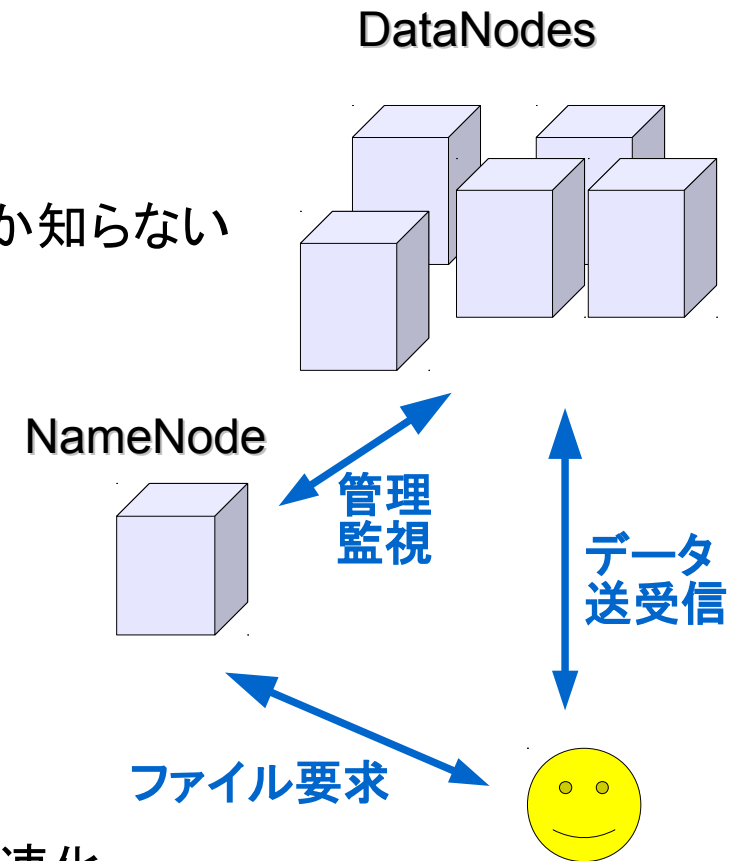


ファイル2



HDFSの構成

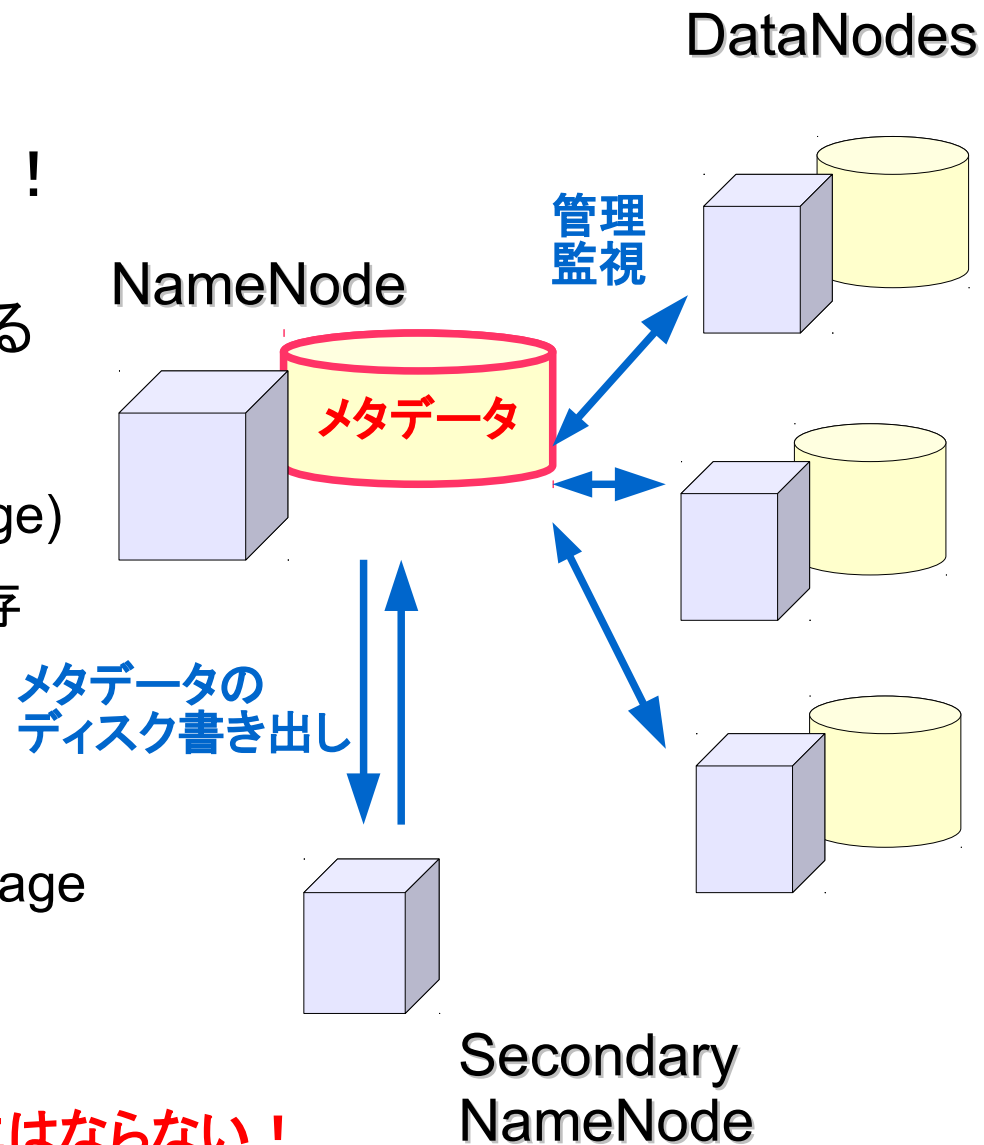
- DataNode (スレーブノード)
 - ファイルを構成するブロックを保存
 - 保持しているブロックがどのファイルを構成するか知らない
- NameNode (マスターノード)
 - クライアントからのファイルの要求に応答
 - メタデータを保持
 - どのファイルがどのブロックで構成されるか
 - どのブロックがどのノードにあるか
 - メタデータは基本的にメモリ上で管理し処理を高速化
 - 実際のデータの読み書きはクライアントと DataNode が直接通信
 - NameNode はボトルネックになりにくい



HDFS の単一故障点

マスタサーバの保護はしてくれない！

- NameNodeが故障するとHDFSが壊れる
 - HDFSメタデータはオンメモリで管理
 - メタデータ全体のスナップショット(fsimage) + 更新差分ログ(edits) をディスクに保存
- SecondaryNameNode
 - fsimage と edits をまとめて新しい fsimage を生成(チェックポイント生成)
 - ⇒ 複数ディスクに書き出し可能
 - **NameNode が故障してもその替わりにはならない！**



HDFS の耐障害性 ～データ損傷への対応～

- レプリケーション
 - ファイルを構成するブロックはデフォルトで3つ複製される
 - 1つのDataNode が故障してもデータが失われることはない
 - データの読み書きの途中に DataNode で失敗しても、他の DataNode に処理が引き継がれる
 - ⇒ クライアント側ではエラーの処理をする必要がない
- DataNode は NameNode に定期的にハートビートを送っている
 - 一定時間応答がない DataNode には障害が発生したとみなされる
 - 既定のレプリケーション数になるまで自動的に複製が行われる
- チェックサム
 - 各ブロックはチェックサム値を持ち、データ損傷の認識と修復が自動的に行われる

Hadoop の並列分散処理 MapReduce処理～概要

- 例えば・・・
 - 右のようなデータ(ユーザID, 行動)から
“login” したユーザの一覧が欲しい場合
- コマンドで実現するならば・・・

```
# usrid  action
00032    login
00015    write_file
00022    logout
00015    read_file
        :
```

```
$ cat input.txt | grep "login" | sort | uniq > output.txt
```

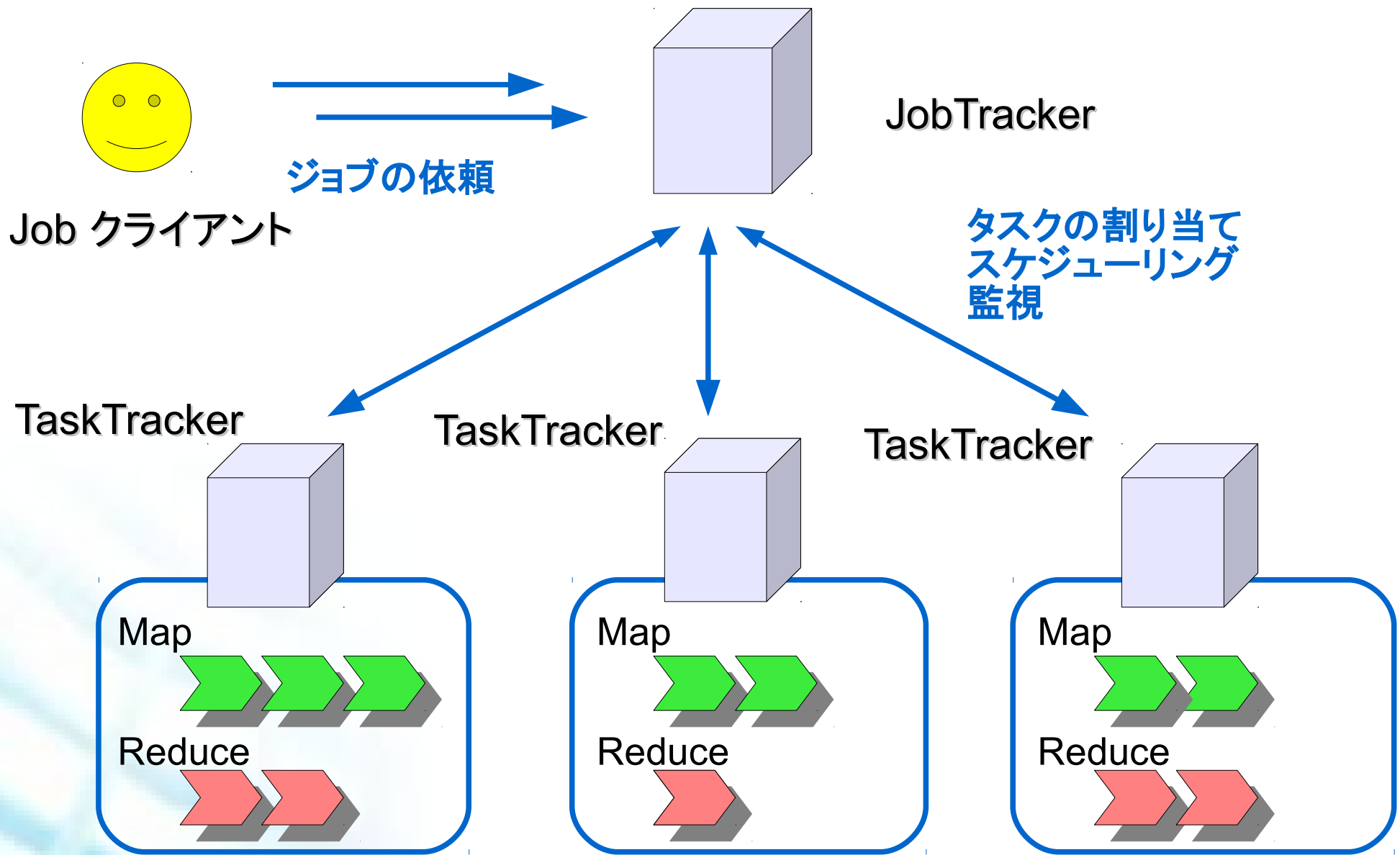


Map

Shuffle & Sort

Reduce

- Map : 処理すべき値の生成
 - Shuffle & Sort : 関連する値を一箇所に集めてソート
 - Reduce : 集められた値の処理
- この Map 処理と Reduce 処理を複数サーバで並列に行う



Map / Reduce それぞれの処理を行うプロセスの生成と実行・管理

Hadoop の並列分散処理 MapReduce処理～詳細

例) 転置インデックス

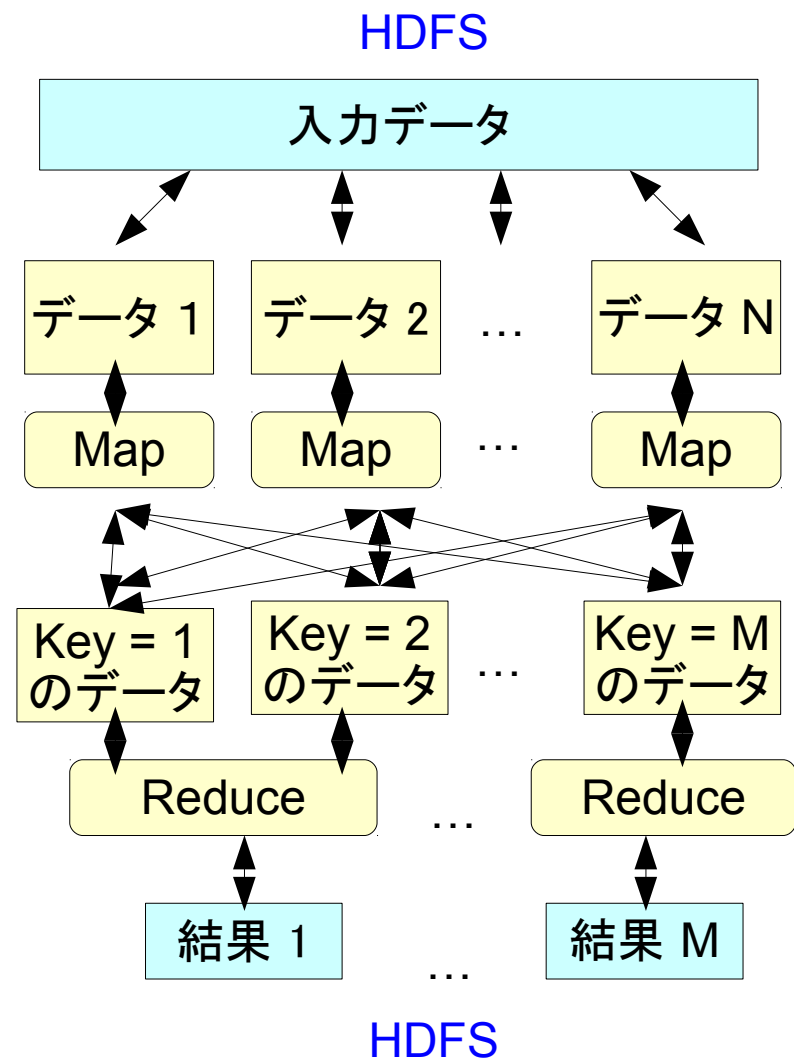
《行番号, その行のテキスト》

↓
《単語, 単語が現れる行のリスト》

- Mapper
 - Key-Value ペアからなるレコードを入力として受け取る
 - 新しく Key-Value ペアを生成して出力 (中間データ)
 - 例) 《行番号, テキスト》 ⇒ 《単語, 行番号》
- Shuffle & Sort
 - 同じ中間 key に関連つけられた全ての中間データが集められる
 - 同じ中間 key を持つデータは全て同じ Reducer に渡される
 - Reducer に渡される際には、Key-Value リストはそのKeyの順序でソートされる
- Reducer
 - Key-Value のリストを入力として受け取る
 - Key の値に関してデータの集計を行い、Key-Value ペアを出力
 - 例) 《単語, 行番号のリスト》 ⇒ 《単語, 行番号リスト(カンマ区切り)》

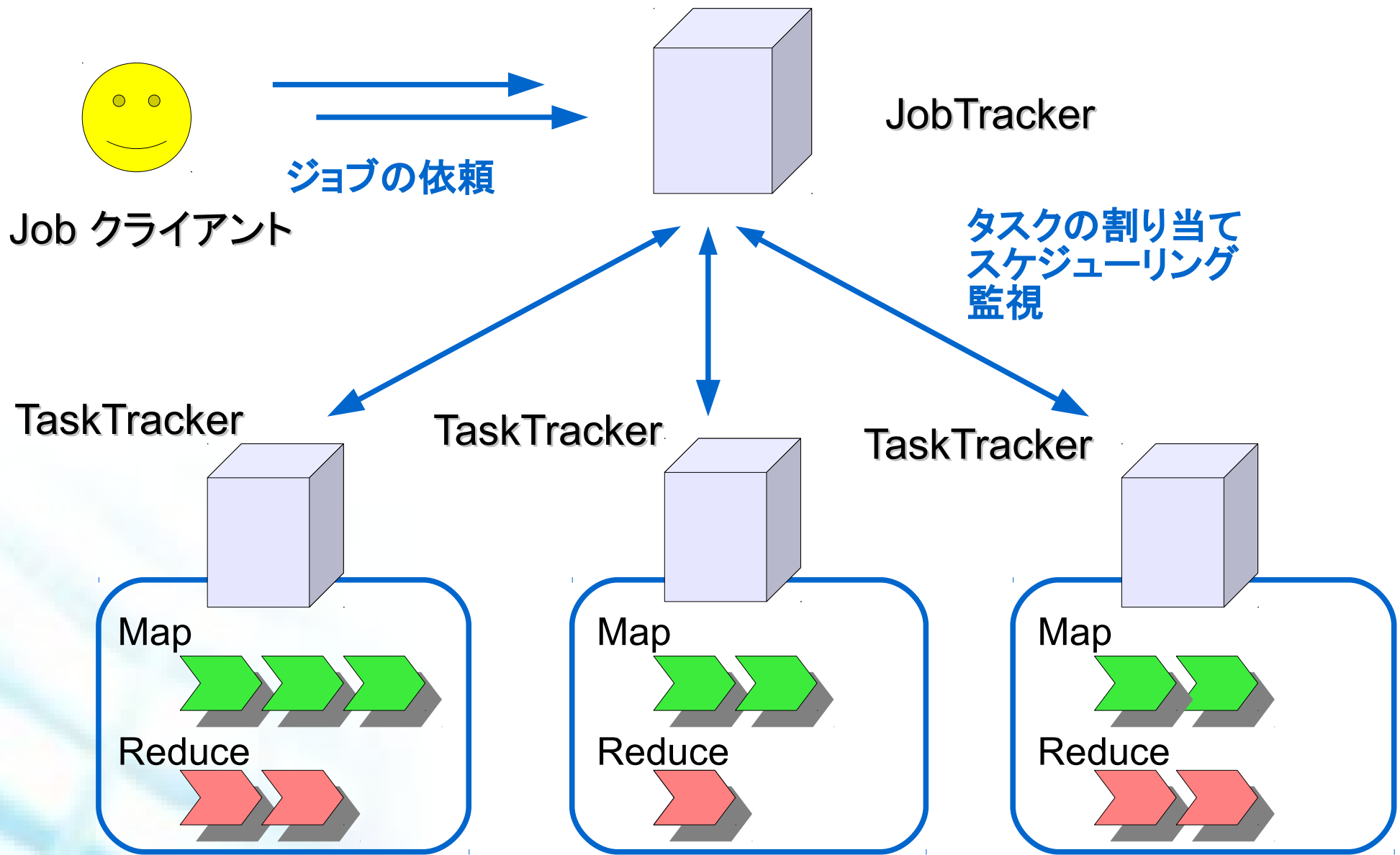
Hadoop MapReduce のフレームワーク

- HDFS 上のファイルが入力
- Mapper : 開発者が記述
 - 入力:Key-Value ペア, 出力:Key-Value ペア
- Shuffle& Sort
 - Keyに従ってデータを Reducer に渡す
 - デフォルトでは Key のハッシュ値を用いる
 - このとき大量のネットワークトラフィックが発生 !
 - Key でソート
- Reduce : 開発者が記述
 - 入力:Key-[Value list], 出力:Key-Value
- 出力はHDFS上のファイル
 - Reducer の数だけファイルが生成される



MapReduceの構成

- JobTracker (マスタサーバ)
 - 依頼された「ジョブ」を「タスク」に分割し、TaskTracker への割り当て & スケジューリングを行う
 - タスクの単位
 - 1つのタスクが1つの入カスプリット(デフォルトではブロック単位)を処理
 - 可能な限り「処理すべきデータを持っているノード」へタスクを割り当てる
 - データがあるところにプログラムを持ってい = **データローカリティ**
 - 単一障害点
- TaskTracker (スレーブサーバ)
 - 割り当てられたタスクの実行
 - Map プロセス、Reduce プロセスの生成・実行・管理
 - JobTracker への進捗報告



Map / Reduce それぞれの処理を行うプロセスの生成と実行・管理

MapReduce の耐障害性 ～部分障害への対応～

- TaskTracker は JobTracker に定期的にハートビートを送っている
- 投機的実行
 - ある TaskTracker が明らかに遅い場合、同じタスクを別の TaskTracker に依頼
 - 処理の完了が早かった TaskTracker の結果を採用する
- 一定期間応答のない TaskTracker は強制終了され、同じタスクが別の TaskTracker に割り振られる
- 一定時間に多数の失敗を起した TaskTracker はブラックリストに登録される
- 1つのタスクが失敗した場合でも、全体のやり直しをせずにジョブを継続可能
 - ただし、タスクが4回(デフォルト)失敗した場合には、ジョブ全体が失敗したとみなされ、MapReduceジョブは終了

MapReduce プログラムの記述

- MapReduce APIの提供
 - 並列分散処理のフレームワーク、障害処理、等を意識する必要がない
 - Map, Reduce の処理を記述するだけ
- 記述は基本的に Java で行う
 - Mapper クラス
 - MapReduceBase クラスを継承
 - Mapper インタフェース (map メソッド) を実装
 - Reducer クラス
 - MapReduceBase クラスを継承
 - Reducer インタフェース (reduce メソッド) を実装
 - ドライバ クラス
 - Mapper クラス, Reducer クラス, 入出力ファイル、形式の指定などジョブの設定
 - ジョブサブミット
 - その他: Combiner, Partitioner, Comparator

MapReduce プログラムの記述

- ジョブの実行

- hadoop jar コマンドを使って、jar ファイルと引数を指定

```
$ hadoop jar wc.jar WordCount inputfile outputfile
```

- Java 以外の方法

- HadoopストリーミングAPI

- 標準入出力を介して map 処理、reduce 処理を行う
 - より慣れた使い慣れた言語、ライブラリ
 - Ruby, Perl, Python, bash, ...

- Hive, Pig

- SQL ライクな言語を用いてMapReduce処理を記述できる

監視のためのWebUI

- WebUIから状況確認と各ノードのログ確認

NameNode 'ossipc24-1.sra.co.jp:54310'

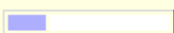
Started: Mon Dec 12 17:52:17 JST 2011
Version: 0.20.205.0, r1179940
Compiled: Fri Oct 7 06:20:32 UTC 2011 by hortonfo
Upgrades: There are no upgrades in progress.

[Browse the filesystem](#)

[Namenode Logs](#)

[Go back to DFS home](#)

Live Datanodes : 2

Node	Last Contact	Admin State	Configured Capacity (GB)	Used (GB)	Non DFS Used (GB)	Remaining (GB)	Used (%)	Used (%)	Remaining (%)	Blocks
ossipc24-2	0	In Service	6.79	0.11	4.27	2.41	1.68		35.5	1139
ossipc24-3	1	In Service	6.79	0.11	4.79	1.88	1.68		27.74	1139

This is [Apache Hadoop](#) release 0.20.205.0

ジョブ管理

- ジョブの過去履歴や進捗状況の把握

Hadoop job_201112121752_0023 on pm01

User: hadoop

Job Name: select count(user) AS frequency, quer...desc(Stage-1)

Job File: hdfs://pm01:54310/tmp/hadoop-hadoop/mapred/staging/hadoop/.staging/job_201112121752_0023/job.xml

Submit Host: pm01

Submit Host Address: 133.137.176.121

Job-ACLs: All users are allowed

Job Setup: [Successful](#)

Status: Running

Started at: Thu Dec 15 18:58:34 JST 2011

Running for: 52sec

Job Cleanup: Pending

Kind	% Complete	Num Tasks	Pending	Running	Complete	Killed	Failed/Killed Task Attempts
map	100.00% 	2	0	0	2	0	0 / 0
reduce	66.68% 	1	0	1	0	0	0 / 0

	Counter	Map	Reduce	Total
File Input Format Counters	Bytes Read	43,667,475	0	43,667,475

Hadoop 周辺ツール

- MapReduce プログラミング
 - Hive
 - Pig
- RDBMSデータの利用
 - Sqoop
- ワークフロー管理
 - Oozie

HiveとPig

- SQL ライクな言語でデータ操作を記述
 - 自動的にMapReduce処理に変換され並列計算を行う
 - Apache License
- Hive
 - HiveQL: 宣言型



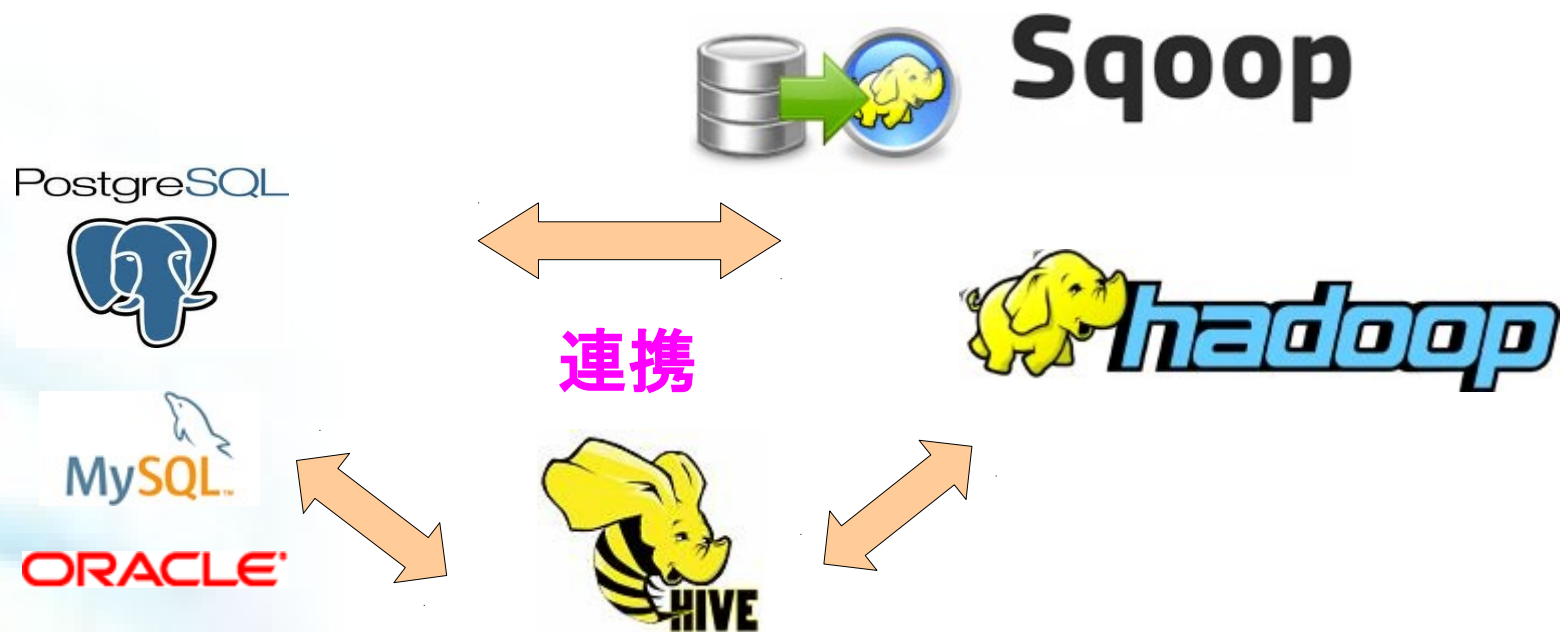
```
hive> INSERT OVERWRITE TABLE pv_users
hive>   SELECT pv.*, u.gender, u.age
hive>   FROM user u FULL OUTER JOIN page_view pv ON
                                           (pv.userid = u.id)
hive>   WHERE pv.date = '2008-03-03';
```

- Pig
 - PigLatin: 手続き型

```
grunt> A = load '/etc/passwd' using PigStorage(':');
grunt> B = foreach A generate $0 as id;
grunt> dump B;
```

Sqoop

- Sqoop = SQL to Hadoop
 - RDBMSからデータをインポート、Hadoop へエクスポート
 - Hiveへのエクスポートも可能
 - Apache License 2.0

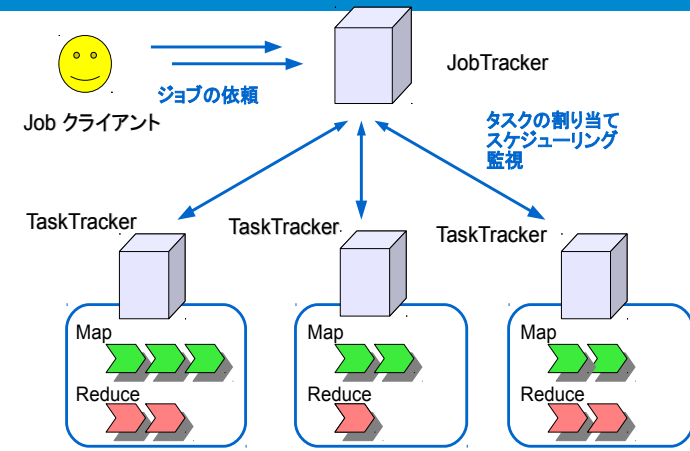
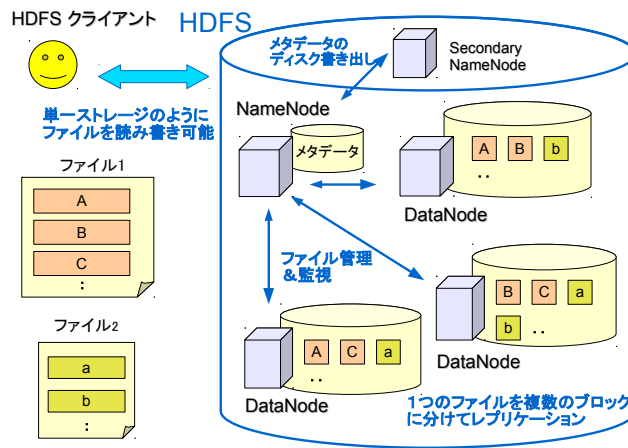


Oozie

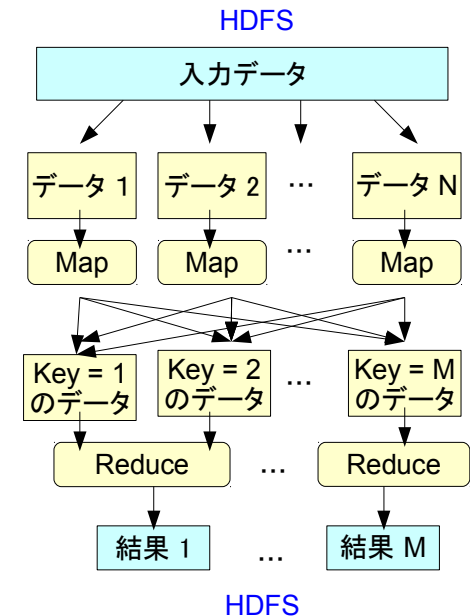
- 複雑な処理を行うには、複数の MapReduce をジョブを多段に行う必要がある
 - 例) ジョブAとジョブBの出力をジョブCの入力とする
- Oozie : ワークフローエンジンThe Oozie logo consists of five blue rounded rectangular blocks with yellow circles, connected by blue arrows pointing from left to right. The blocks contain the letters 'O', 'O', 'Z', 'i', and 'E' respectively. A small 'TM' trademark symbol is at the top right of the 'E' block.
- Apache License 2.0
- Hadoop のワークフローを実行
- ワークフローは XML で記述する
- ジョブには Hive, Pig, Sqoop を含めることもできる

Hadoop まとめ

- HDFS
 - 分散ストレージ
- MapReduce
 - 並列計算フレームワーク
- 障害に強いシステム
 - データが失われない
 - 処理が途中で止まらない
 - システムが自律的に障害から復旧
- Hadoop フレームワークがこれらの面倒を見てくれる
⇒ 開発者はMapReduce処理の開発に専念できる



Map / Reduce それぞれの処理を行うプロセスの生成と実行・管理



OSSプロフェッショナルサポートサービス

33種類以上の幅広いOSSをワンストップでサポート。台数無制限のサービスです。

サービス内容

ヘルプデスク

障害対応
(プロメニューのみ)

ナレッジサービス

情報配信サービス

サービス対象ソフトウェア

FTPサーバ
ProFTPD, vsftpd

キャッシュサーバ
Squid

ロードバランサ/
リバースプロキシ
Pound

DNSサーバ
Bind

ファイル/プリントサーバ
Samba

LDAPサーバ
OpenLDAP

メールサーバ
Postfix, sendmail
qmail, Dovecot
UW-IMAP
Courier-IMAP
Qpopper

運用監視: Hinemos
Zabbix

KVS: memcached
Kyoto Cabinet, Kyoto Tycoon

分散処理: Hadoop

シングルサインオン
OpenAM

Webサーバ: Apache

APサーバ: Tomcat

DBサーバ: PostgreSQL
SQLite

HAソフトウェア: Heartbeat, Pacemaker, DRBD

仮想化: Xen, KVM

OS: CentOS

ご清聴ありがとうございました



アンケート よろしくお願ひ致します。

MapReduce 例: ワードカウント

- $\langle \text{Key} = *, \text{Value} = \text{テキスト} \rangle \Rightarrow \langle \text{Key} = \text{単語}, \text{Value} = \text{単語出現回数} \rangle$

```
Map (Key, Value) {  
    foreach (word in Value) {  
        output (word, 1);  
    }  
}
```

テキストを「単語」に分割して、
「単語」と整数「1」を出力

```
Reduce (Key, Values) {  
    count = 0;  
    foreach (value in Values) {  
        count += value;  
    }  
    output (Key, count);  
}
```

各単語について、出現回数を合計

MapReduce 例: 転置インデックス作成

- ・ <Key = 行番号, Value = テキスト> ⇒ <Key = 単語, Value = 単語を含む行番号リスト>

```
Map (Key, Value) {  
    foreach (word in Value) {  
        output (word, Key);  
    }  
}
```

テキストを「単語」に分割して、
「単語」と「行番号」を出力

```
Reduce (Key, Values) {  
    list = list (Values);  
    output (Key, list);  
}
```

各「単語」について「行番号のリスト」を出力

WordCount: Mapper

```
public static class Map extends MapReduceBase
    implements Mapper<LongWritable, Text, Text, IntWritable> {
    private final static IntWritable one = new IntWritable(1);
    private Text word = new Text();

    public void map(LongWritable key, Text value,
        OutputCollector<Text, IntWritable> output, Reporter reporter)
        throws IOException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            output.collect(word, one);
        }
    }
}
```

入出力 Key-Value の型を指定

map メソッドの実装

Hadoop の「型」

IntWritable
LongWritable
Text
:

WordCount: Reducer

```
public static class Reduce extends MapReduceBase
    implements Reducer<Text, IntWritable, Text, IntWritable> {
    public void reduce(Text key, Iterator<IntWritable> values,
        OutputCollector<Text, IntWritable> output, Reporter reporter)
        throws IOException {

        int sum = 0;
        while (values.hasNext()) {
            sum += values.next().get();
        }
        output.collect(key, new IntWritable(sum));
    }
}
```

入出力 Key-Value の型を指定**reduce メソッドの実装**

WordCount: ドライバ

```
public class WordCount {  
    public static void main(String[] args) throws Exception {  
        JobConf conf = new JobConf(WordCount.class);  
        conf.setJobName("wordcount");  
        conf.setOutputKeyClass(Text.class);  
        conf.setOutputValueClass(IntWritable.class);  
        conf.setMapperClass(Map.class);  
        conf.setCombinerClass(Reduce.class);  
        conf.setReducerClass(Reduce.class);  
        conf.setInputFormat(TextInputFormat.class);  
        conf.setOutputFormat(TextOutputFormat.class);  
        FileInputFormat.setInputPaths(conf, new Path(args[0]));  
        FileOutputFormat.setOutputPath(conf, new Path(args[1]));  
        JobClient.runJob(conf);  
    }  
}
```

ジョブの設定

出力 Key-Value の指定

Mapper, Combiner, Reducer の指定

入出力形式の指定

入出力ファイルパスの指定

ジョブのサブミット