衛星データ処理勉強会
分散処理システムHadoop

宇宙航空研究開発機構
ISAS 宇宙科学情報解析研究系/JSPEC 研究開発室
山本 幸生

クラウドとは?
人によって「クラウド」という言葉の意味が違う

- **インターネットの先にあるシステムという意味で使う人**
  - ネットワーク図でよくインターネットを表すときに雲の図が使われ、インターネットの先には何か便利なものがあって、それらがクラウドでできている、と主張
- **サービスのことを言う人**
  - SaaS (Software as a Service)
    - エンドユーザ向けのサービス(ウェブメール,ブログ etc.)
  - PaaS (Platform as a Service)
    - サービス提供者向けのサービス(Google App Engine, セールスフォース・ドットコム, Mixi Appli etc.)
  - IaaS (Infrastructure as a Service)
    - レンタルサーバを仮想マシンで置き換えたようなサービス(Amazon EC2)
- **ハイパーバイザ型仮想化システムのことを言う人**
  - VMWare ESX, Hyper-V, Xenなどハードウェア構築の負担を減少させた仮想化システム (弾力性と言う人も)
  - プライベートクラウド導入しませんか～？という売り文句はだいたいこの仮想化システムのこと
- **グリッドコンピューティングの別名と主張する人**
  - グリッドコンピューティングとは複数の場所に散在するコンピュータをネットワーク上でつなげてリソース(CPU,ネットワーク,ストレージetc.)を共有する仕組み
  - 実際にクラウドと呼ばれるハードウェアやソフトウェアの機能はグリッドコンピューティングと区別がつけにくい
- **一時的に不完全な状態を許したのがクラウドと主張する人**
  - 1万円をA銀行からB銀行に振り込む処理で「A銀行の1万円減額」「B銀行の1万円増額」を同時にやるのがこれまでのシステム
    途中で「B銀行の1万円増額」が遅れても最終的に同じであればよしと考え、
    分散に伴うリスクとして遅延があることを前提に設計されているのがクラウド

お互いが想像する意味が違ってもだいたい会話が通じる。なぜ？

⇒「大規模な処理をこなせるシステム」という意味で意見が一致

実はよく分かってない「もやもや」したもの

Hadoopはクラウドだ！

- 3つの概念から構成
  - 分散ファイルシステム(Hadoop Distributed File System; HDFS)
    - 複数のマシン上でデータを共有し冗長化するためのファイルシステム。ネットワーク上で構成されたRAIDのようなもの。FUSEを使ってmountすると一般のファイルシステムとほぼ同等に扱える。
  - 分散処理(Map/Reduce)
    - 複数のマシン上で処理を分散し、分散処理された結果を最後に集めて集計する。
  - 分散データベース(hBase/HyperTable)
    - Google BigTableのようにカラム型データベースを実現する。
- 上記のどれかにだけ注目して利用することも可能
  - RAIDの代わりにHDFSのみ利用してデータの冗長化をしたい
  - 分散処理だけ利用したい
    （※ Hadoopで分散処理するにはHDFS上にファイルを置いた方が分散性があがる)
  - HDFSを使わずカラム型データベースだけ利用したい

結局Hadoopで何ができるの？という問いには…

データの冗長化ができる

Google BigTableのようなものを自社環境で利用できる(改造もOK)

分散処理を少ない手間で記述・実行できる
（分散性の高い処理に限る)

汎用的なPC数台で構築できる

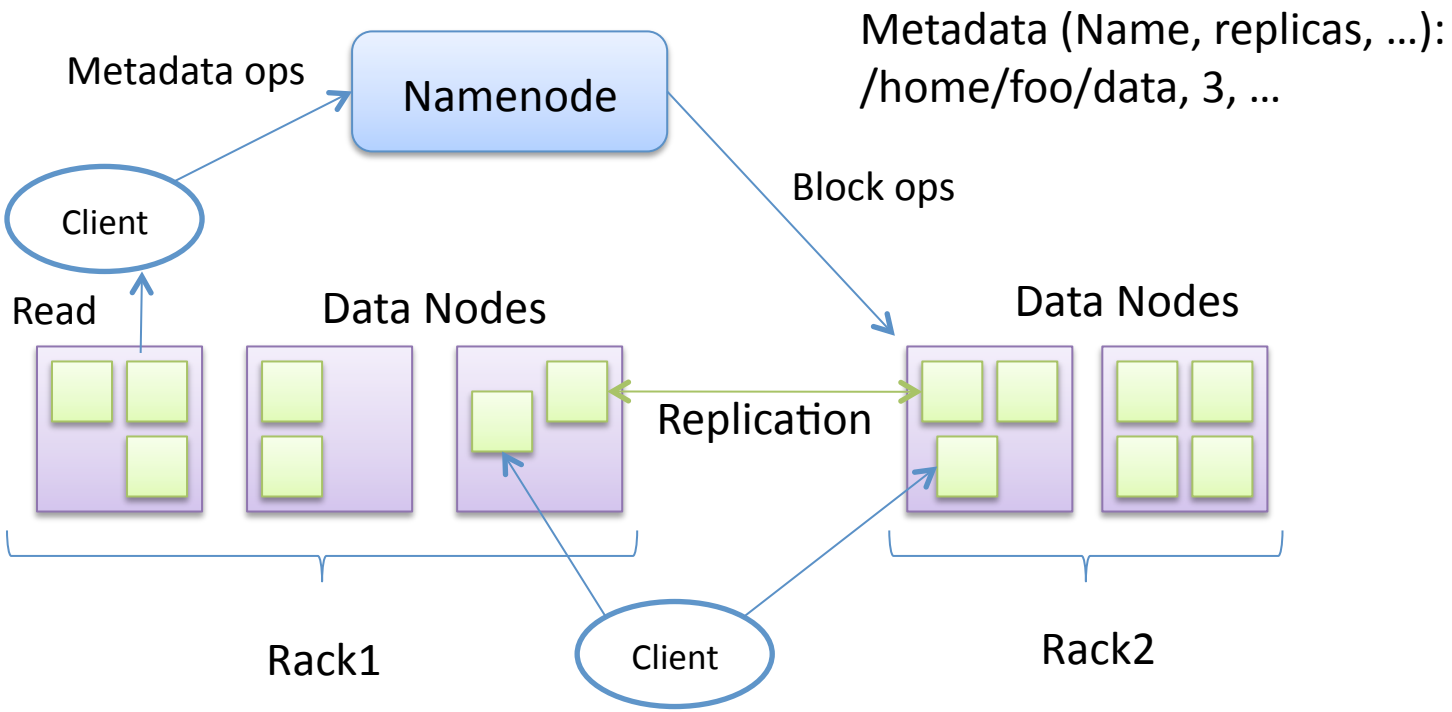
※ 構築・運用コストは安くないので自前で構築する場合は要注意

分散ファイルシステムとしてのHadoop
Hadoop Distributed File System (HDFS)

HDFSの特徴

- 特徴
  - Write Once/Read Manyのアクセスモデルに最適化
  - アクセスパーミッション (POSIXライク)
    - ownerとgroupにread/write/execを付与
    - sticky bit (/tmpのように誰でも書きこめて所有者や管理者のみ削除可能な機能)はなし
    - setuid,setgid (実行時に所有者の権限で実行する機能)はなし
  - クォータを設定可能
    - Name Quotas … ファイルとディレクトリの最大数
    - Space Quotas … 1ファイルで扱える最大サイズ
- 利点
  - 超巨大ファイルに対応
    - 数GB～数PBのファイルでも格納可能
  - 安価なハードウェアで高可用性を実現
    - 複数のハードウェアで運用することを前提に作られているため1台くらい壊れても問題なし
    - ファイルのコピーは複数のシステムに分散されている
- 欠点
  - Read/Writeが遅い
    - 特にWriteはかなり遅い
  - 大量の小さいファイルは苦手
    - NameNodeのヒープメモリが枯渇する
  - ディスクの使用効率が悪い
    - RAIDのようにパリティのみ格納というモデルではなくそのまま別ハードウェアに保存
  - NameNodeとして利用するサーバは高可用性が必要

HDFSのアーキテクチャ



Cf. The Hadoop Distributed File System: Architecture and Design
http://hadoop.apache.org/common/docs/r0.18.0/hdfs_design.pdf

HDFSの信頼性は?

- ハードウェア障害 (ディスク障害、ネットワーク障害など)
  - Datanodeに障害発生
    - データを複数持つので問題なし
    - レプリケーションレベルRの分だけ耐久性あり(デフォルトは3)
    - レプリカが破損すると自動修復 (ネットワークトラフィックが増加するので台数が少ないときは高負荷に注意)
  - Namenodeだけは単一障害ポイント
    - Namenodeにトラブルが発生するとディスク全体が利用できなくなるので高可用性のシステム構築が必要
    - Secondary NamenodeがあるがPrimary Namenodeのスナップショットを定期的にダウンロードするだけでPrimary Namenodeハングアップ時のバックアップとして動作する訳ではない

Cf. HDFSの信頼性はHDFS Reliability, Tom White, Cloudera, 12 January 2008に整理されている

HDFSと他ストレージの単純比較

- 科学データを格納しているNetAppやテープドライブをHDFSで代替できるか?
  - Read/Writeの速度を無視すればできる
    - NetAppの代用は難しいがテープドライブの代用は比較的現実的 (ただし運用実績が欲しい)
- コスト見積もり・設置面積はどのくらい?
  - SANやNASと違いストレージのみ巨大化という訳にはいかないため単純な比較は無意味
  - とは言え一般的なストレージと比較してみたい

製品名	数量	物理容量	実効容量	設置面積	電力	価格	参考源泉
NetApp FAS6080	1,176台 (HDD数)	1,176TB (1TB/台)	784TB (RAID-4)	256U, 6-8 racks (84 shelves , 3U/shelf)	< 40 kW	～10億円未満	
NTC Supremacy	448台 (HDD数)	896TB (2TB/台)	712TB (RAID-6)	84U, 2-4 racks (7台x3Ux4)	<15 kW	～1億円未満	NTC ウェブサイト
DELL PowerEdge R200	1,176台	2,352TB (2TB/台)	784TB (HDFS,R=3)	1,176U, 28-42 racks	<400 kW	～2億円未満 (15万円以下 x 台数)	DELL ウェブサイト
DELL PowerEdge M610 (Blade)	2,352台	2,352TB (1TB/台)	784TB (HDFS,R=3)	1,470U, 37 racks (M1000e x 147)	<800 kW	～6億円未満 (25万円以下 x 台数 +エンクロージャ)	DELL ウェブサイト
ATOM Server	1,176台	2,352TB (2TB/台)	784TB (HDFS,R=3)	1,176U, 28-42 racks	<200 kW	～2億円未満	

※ NetAppは数年前の価格のため参考値。他は2009年3月の価格。

ストレージだけを考えた場合市販のストレージ製品を購入した方が良い。設置面積・電力はHDFSより格段に優れている。またストレージとしての速度・機能・運用性・実績も充実しており、さらに価格については信頼性に応じた製品選択が可能である。
Hadoopで「分散処理の実施」を含めて考えた場合は対象とするファイルがHDFS上にあった方が処理の分散性能が良い。
またHDFSはパーティションの境界がないため「巨大なファイル(1PB等)を作成」する場合に有用となる。

分散処理システムとしてのHadoop

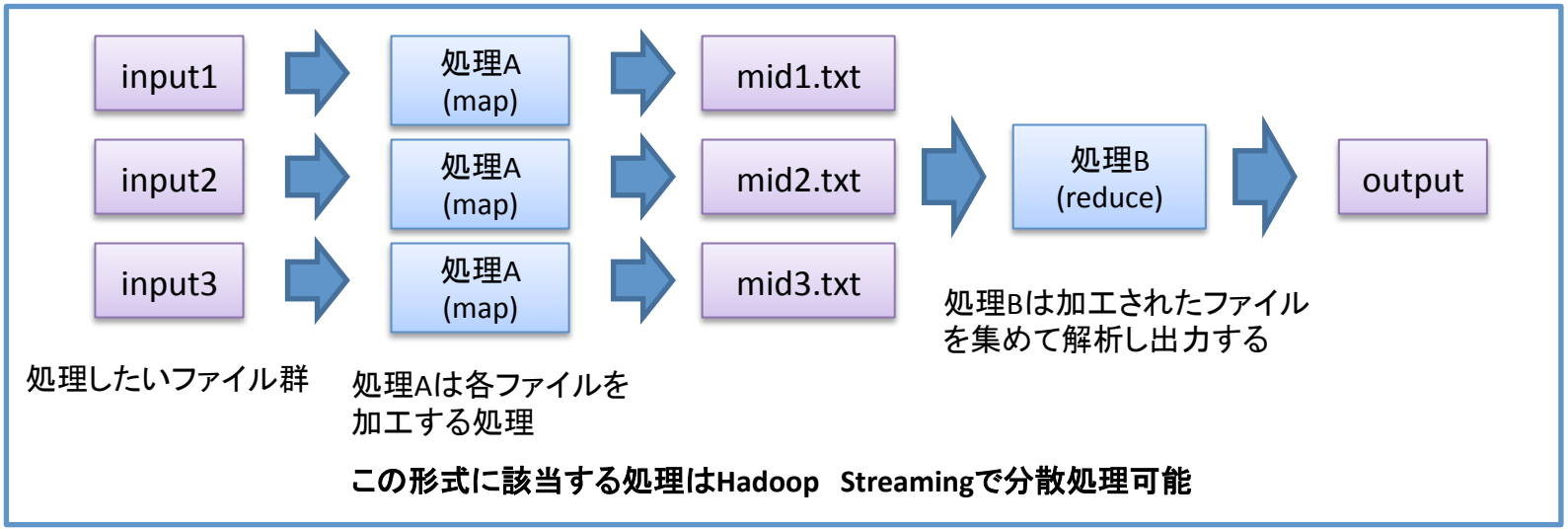
Map/Reduce Algorithm

MAP/REDUCEアルゴリズム

- Hadoop Streamingを使うと概念を掴みやすい
  - Hadoop Streamingは標準入出力を利用してMap/Reduceを実現する(inetdに似ている)
  - 例えば次のスクリプトで書けるものはHadoop Streamingですぐに処理できる

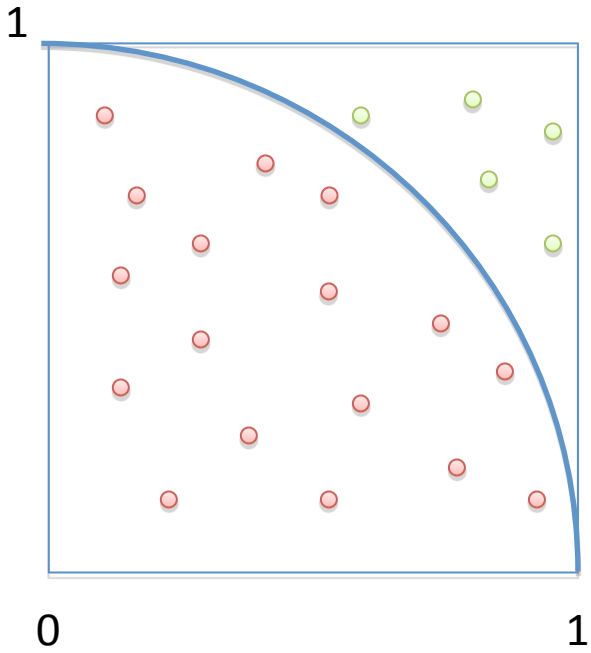
```
$ cat input1.txt | map.sh > mid1.txt
$ cat input2.txt | map.sh > mid2.txt
$ cat input3.txt | map.sh > mid3.txt
$ cat mid1.txt mid2.txt mid3.txt| reduce.sh > output.txt
```

- ポイント
  - mid1, mid2.txt, mid3.txtの順序を入れ替えても正しく動作するようにreduce.shを作る
  - 中間ファイルであるmid1.txt等は標準では取り出すことはできない(reduceをなくせば取り出せる)
  - 入力ファイルはHDFS上に置く必要があり、また出力ファイルもHDFS上に置かれる
  - 外部ファイルは基本読み込めない (読み込みたい場合は一か所に集めてjarファイルを作る必要あり)



Hadoop処理の例1

MapReduceによる円周率計算1



モンテカルロ計算による円周率の求め方
x: 0から1までの乱数
y: 0から1までの乱数
として、点(x,y)をランダムに生成する。

・(x,y)が円弧の中に含まれた場合の数をinside
・(x,y)が円弧の外に含まれた場合の数をoutside
とすると扇形の面積はπ/4、正方形の面積は1なので、
扇形の中の点数/全体の点数=π/4
が成立する。

すなわちinside/(inside+outside)=π/4となるので、πを求めるには4*inside/(inside+outside)を計算すればよい。

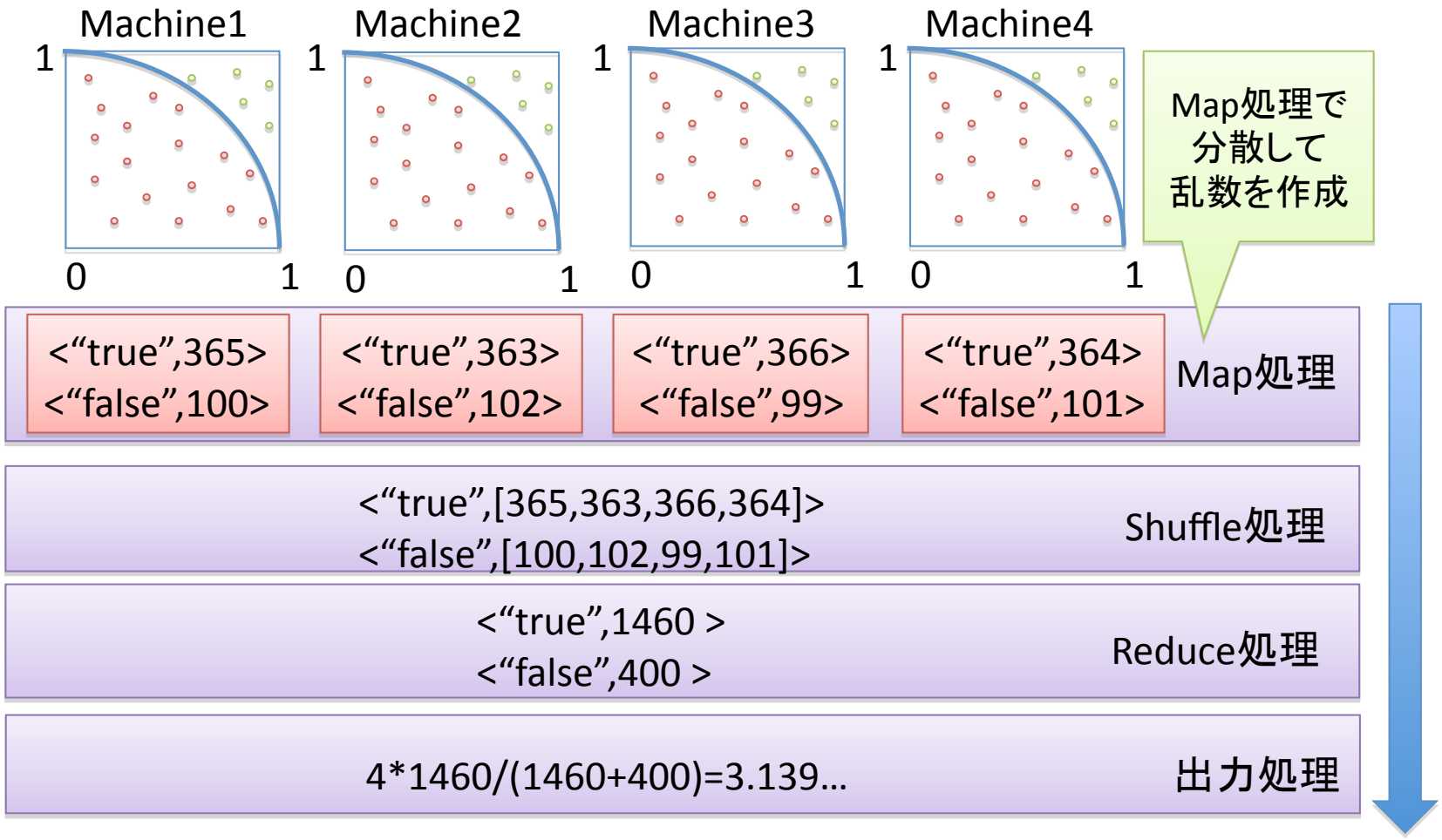
Hadoopを使って円周率を求めるには、乱数を発生させMap処理によって

<“true”, inside>　<“false”, outside>

という<キー,値>のペアを作成する。このマップは分散された個数だけ生成されるので、Reduce処理によって集計する必要がある。
”true”, ”false”というキーに対して集計すると、処理全体でのinside, outsideが求まる。
最後に終了処理として4*inside/(inside+outside)を出力する。

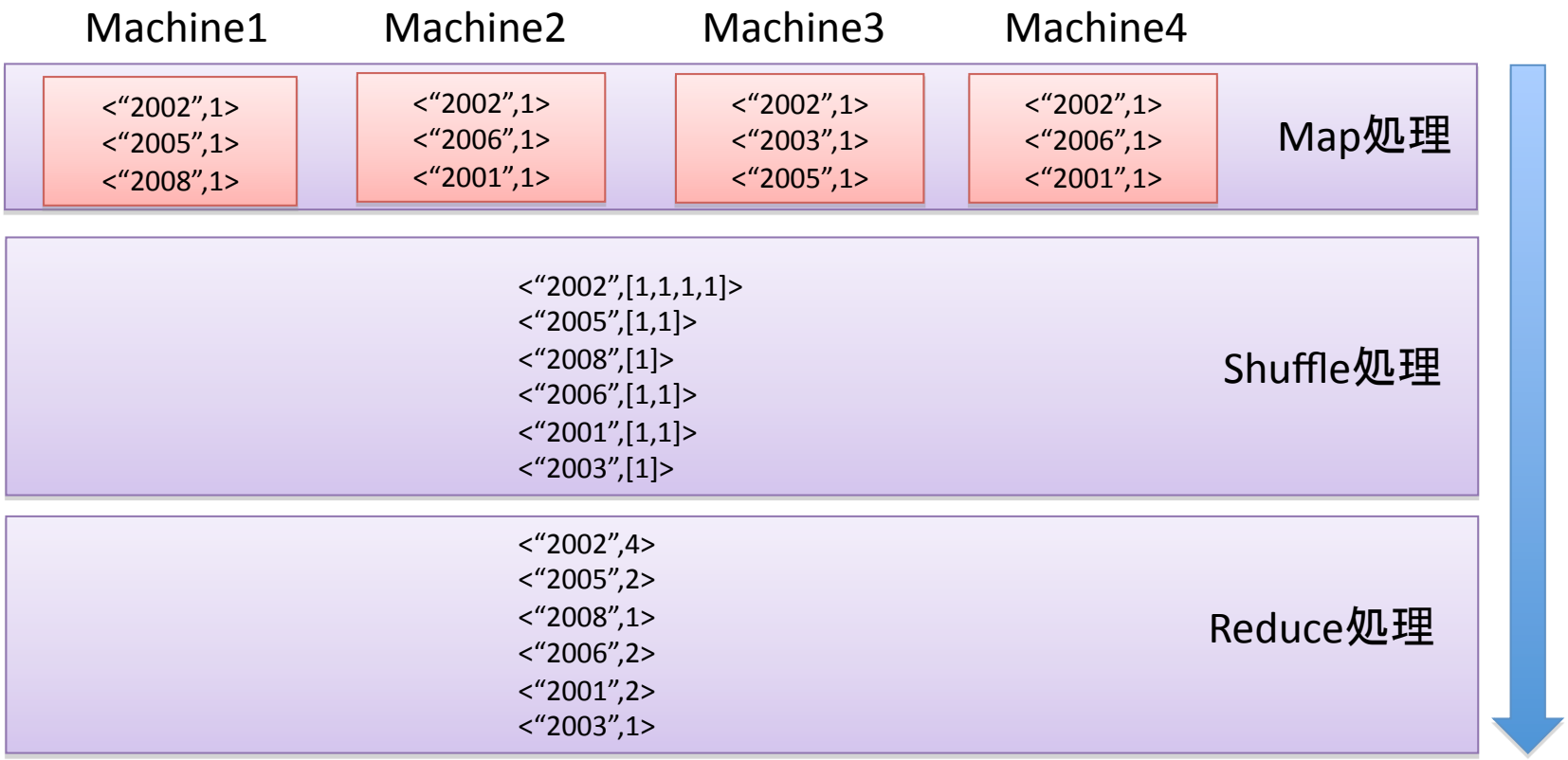
Hadoop処理の例1

MapReduceによる円周率計算2



Hadoop処理の例2
Grep検索

Ex) 2001から2009までの文字列の数をindirディレクトリから探しカウントする
Grep indir outdir 200[1-9]



Hadoop Map処理の入力と出力

入力として「1つの巨大なファイル」でも「複数の小さなファイル」でも等価に扱える。
またファイルの内容に応じて「InputFormatクラス」という入力フィルタを利用し分割可能な単位を決定する。

分割される最小単位に対して、とにかく **<Key,Value>のペアを作ること** を目指す。
この<Key,Value>のペアがmap処理に渡される。
<Key,Value>はその後のReduce処理に渡され、特にKeyはReduce処理を再度分散させるシャッフル処理で利用される。

例1) 巨大なテキストファイル

```
[Sat Oct 03 21:34:25 2009] [error] [client xxx.xxx.xxx.xx]
File does not exist: /var/www/favicon.ico
[Sat Oct 03 21:35:20 2009] [error] [client xxx.xxx.xxx.xx]
File does not exist: /var/www/favicon.ico
[Sat Oct 03 21:35:59 2009] [error] [client xxx.xxx.xxx.xx]
File does not exist: /var/www/favicon.ico
```

TextInputFormat
(1行1レコード)

<行番号,行文字列>

```
<10, [Sat Oct 03 21:34:25 2009] [error] [client xxx.xxx.xxx.xx] File does not exist: /var/www/favicon.ico>
<11, [Sat Oct 03 21:35:20 2009] [error] [client xxx.xxx.xxx.xx] File does not exist: /var/www/favicon.ico>
<12, [Sat Oct 03 21:35:59 2009] [error] [client xxx.xxx.xxx.xx] File does not exist: /var/www/favicon.ico>
<13, [Sat Oct 03 21:36:59 2009] [error] [client xxx.xxx.xxx.xx] File does not exist: /var/www/favicon.ico>
```

例2) 複数のテキストファイル

The quick brown fox
jumps over the lazy dog.
quick waltzes and jigs.
chumps quickly in fog.

TextInputFormat
(1行1レコード)

<文字数,行文字列>

```
<35,The quick brown fox jumps over the lazy dog.>
<36, Heavy boxes perform quick waltzes and jigs.>
<36, A wizard's job is to vex chumps quickly in fog.>
```

例3) 複数のバイナリファイル

バイナリファイル

WhoeFileInputFormat
(1ファイル1レコード)

<なし,バイナリデータ>

```
<null, 1f 8b 08 00 82 d4 c7 4a 00 03 ed 5d 5d 6f db c6...>
<null, 12 7d ef af 58 f4 21 69 81 44 e2 f2 9b 04 82 22...>
<null, 48 dc 8f 8b a6 31 62 07 c5 bd 41 60 28 12 6d 13...>
```

Hadoop Map処理の例

(天文や惑星分野で多そうな)画像を加工する例

画像をまたいだ処理がない場合、Reduce処理は行わない

Map/Reduceで書くと

<null, image data> ---(map) ---> <text,text> ---(reduce) ---> (text,text)

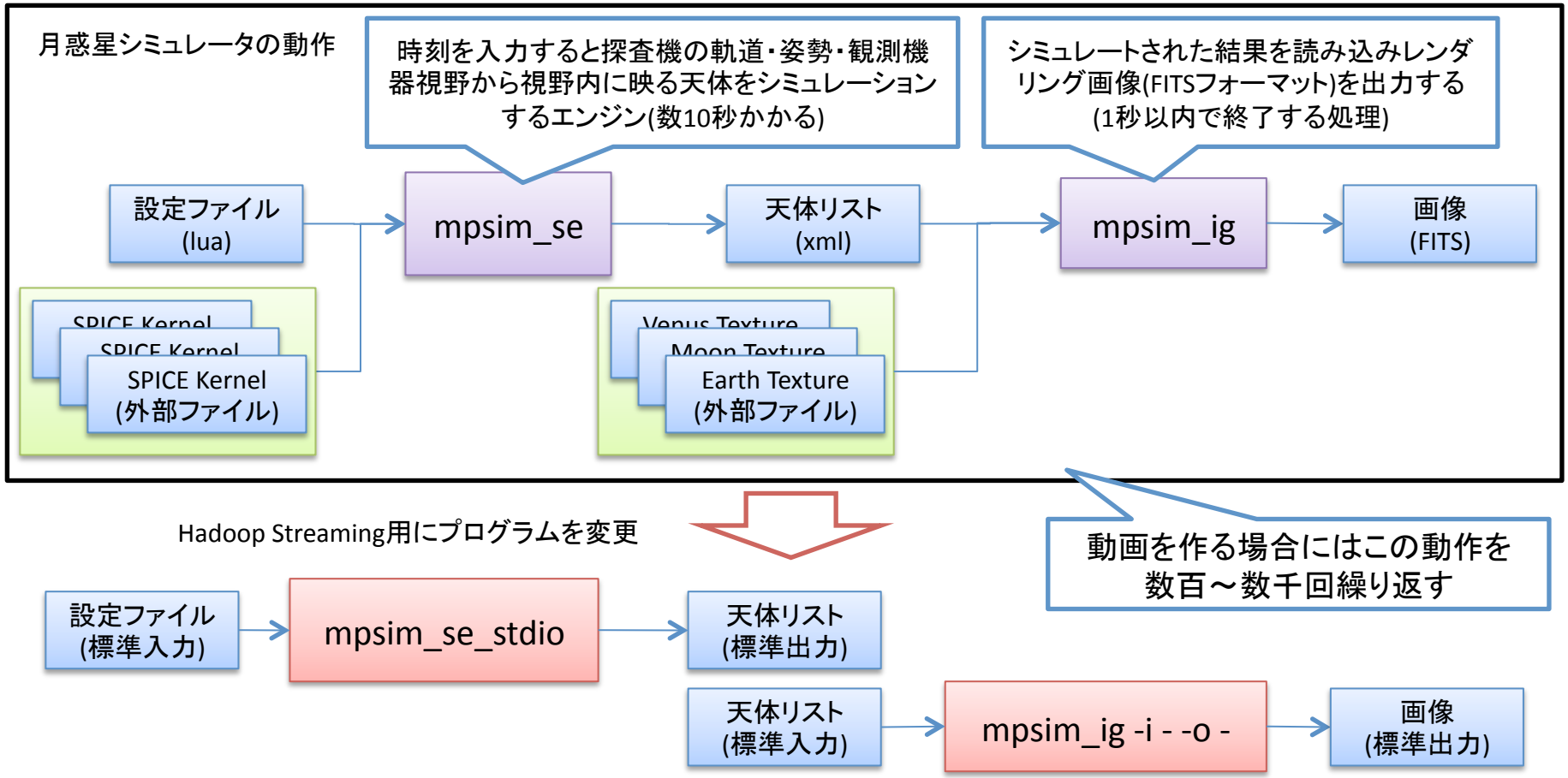
Reduce処理がないのでなんでも良い

Hadoopにおける実際の処理方法

1. <null, image data>を作るために入力フィルタとなるクラスを作成
   # InputFormat継承クラスを作成 (WholeFileInputFormatクラス)
   isSplitableとgetRecordReaderメソッドをOverrideする
   ファイルが途中で分割されないようメソッドisSplitableは戻り値として常にfalseを返す
   getRecordReaderは次のクラスのインスタンスを新規作成して返す
   # RecordReader継承クラスを作成(WholeFileRecordReaderクラス)
   <key,value>としてのうちkeyを捨ててvalueに画像データを入れる(<NullWritable, BytesWritable>のペアを作成)

2. Hadoopによって起動されるクラスを作成
   # map処理で呼び出されるmap関数を作成する(value.getBytes()を呼び出すことで画像データ取得)
   # reduce処理で呼び出されるreduce関数を作成する(今回はなし)
   # main関数で初期設定(呼び出すクラス,使用するInputFormat等)を行いjobを走らせる

Hadoop Streamingを使った月惑星シミュレータの分散化

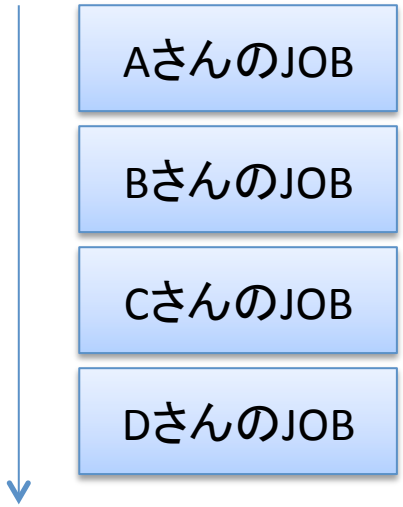


・読み込まれる設定ファイル群は予め用意してhdfs上の同じディレクトリにアップロードする
・プログラムは標準入力と標準出力で動作するように改造する
・外部ファイルとなるSPICE Kernelは一か所に集めjarで固めてhdfs上のどこかにアップロードする
・起動時のオプション-cacheArchiveを指定してファイルパスを設定する
・外部ファイルを読み込む部分のパスを全て変更する(ここでは設定ファイルのパスを変更)

みんなで使うHadoop
ジョブの投入とスケジューラ

複数人でHadoopを利用する場合、スケジューラを適切に選択する必要がある

・FIFO Scheduler
(JOBは投入した順番で処理される)



・Fair Scheduler by Facebook
・Capacity Scheduler by Yahoo
(JOBは公平に処理される)

プリエンプション処理
(実行の中断)機能あり

- FIFO Schedulerでは他のJOBの終了を待機しないと新しいJOBが流れない
- Fair Schedulerでは指定された時間内に終了するJOBを流す必要があり、JOBが終わらない場合は中断し、その後何度かトライして諦めてしまう

Hadoop向きの処理・Hadoop向きでない処理

- Hadoop向きの処理
  - 分散性の高い処理 (単純な繰り返しが多い処理)
    - パラメータサーチ (フィッティング含む)
    - 暗号解読
- Hadoop向きでない処理
  - 分散性の低い処理(他のノード・どこかの処理結果を利用する処理)
    - 時間発展型のシミュレーション
    - 巨大な行列計算
  - 短時間で終わり繰り返す必要のない処理
    - Hadoopのオーバーヘッドは大きい

分散データベースとしてのHadoop

カラム型データベースHBase

key-valueストア

MapReduceアルゴリズムはkey-valueストア型データベースの分散処理に向いている

key-valueストア型

Berkeley DB
GDBM
Memcached
…

Key1	Value1
Key2	Value2
Key3	value3

RDBMS

Oracle
SQL Server
MySQL
PostgreSQL
…

Key	column1	column2	column3
Key1	Value1	Value2	Value3
Key2	Value4	Value5	Value6
Key3	Value7	Value8	Value9

key-valueストアとシリアル化

- key-valueストアでRDBMSのように複数の情報を保持したい場合、複数の情報のシリアル化(直列化)を実施し、valueとして値を格納する
- HadoopではHBaseと呼ばれるカラム型データベースがあるため、それを利用する

key	column1	column2	column3
Key1	Value1	Value2	Value3
Key2	Value3	Value1	Value5

シリアル化

Key	Value
Key1	Value1:Value2:Value3
Key2	Value3:Value1:Value5

カラム型データベース (HBase/HyperTable)の利用

Product	Header:temp	Header:target	Document:content
product1	10.3	Moon	“This document…”
product2	11.0	Mars	“Description:…”
product3	9.8	Moon	“REMARK:…”

Hadoopではカラム型データベースとして HBaseを用意している。また同様の位置づけでC++で記述されたHyperTableも存在する。

テーブルという意味では通常のRDBMSと同様に扱えるが、基本的にKey-Valueストアなので Keyでしか検索できない。概念をよく理解した上で利用することにより、要求に対する最適な設計をすることが可能となる。

またSQLのようなクエリはそのままでは利用できないためプログラムから挿入・検索・削除を利用する。サブプロジェクトとして存在するhiveを用いることでSQLを利用可能である。

カラム型データベース1

Google BigTableやHadoop HBaseはカラム型データベースと呼ばれる。
カラム型データベースはプログラミング言語における連想配列で考えると分かり易い。

key-valueストアの場合

```
{
        “1”:”suzuki”,
        “2”:”sato”,
}
```

カラム型データベースの場合

```
{
        “1”: {
                “name” : “suzuki”,
                “age” : “25”
        },
        “2”: {
                “name” : “sato”,
                “age” : “28”
        }
}
```

“1”や”2”の部分を「行キー」といい、行を決めるために必要で重複してはいけない。
nameやageのことを「カラムファミリー」と言い、各項目で一致している必要がある。
カラムファミリーはデータベース作成時に指定し、後で変更できない。

カラム型データベース2

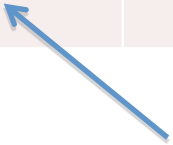
カラムファミリーを変更することはできないが、
カラムファミリーは好きなだけkey,valueのペアを持てる

```
{
    “1”: {
        “profile” : {
            “name” : “suzuki”,
            “age” : “25”,
            “hobby” : “baseball”,
        }
    },
    “2”: {
        “profile” : {
            “name” : “sato”,
            “age” : “28”,
            “job” : “teacher”
        }
    }
}
```

左の例ではカラムファミリーはprofile。
一度作成したら変更は不可能だが、name, age, hobby, job は動的に好きなだけ追加できる

テーブル表記で表すと…

Row key	profile:name	profile:age	profile:hobby	profile:job
1	suzuki	25	baseball	
2	sato	28		teacher



この枠1つのことをCellと呼び、データはCellごとに保持される

カラム型データベース3

それぞれのCellの値はTimestampを記録しバージョンという概念がある

```
{
   “1”: {
      “profile” : {
         “name” : {
            “2000-01-11” : “suzuki”
         }
         “age” : {
            “2000-01-11” : “25”,
            “2005-10-12” : “30”
         }
         “hobby” : {
            “2000-01-11” : “baseball”,
            “2005-10-12” : “computer”
         }
      }
   }
   …
}
```

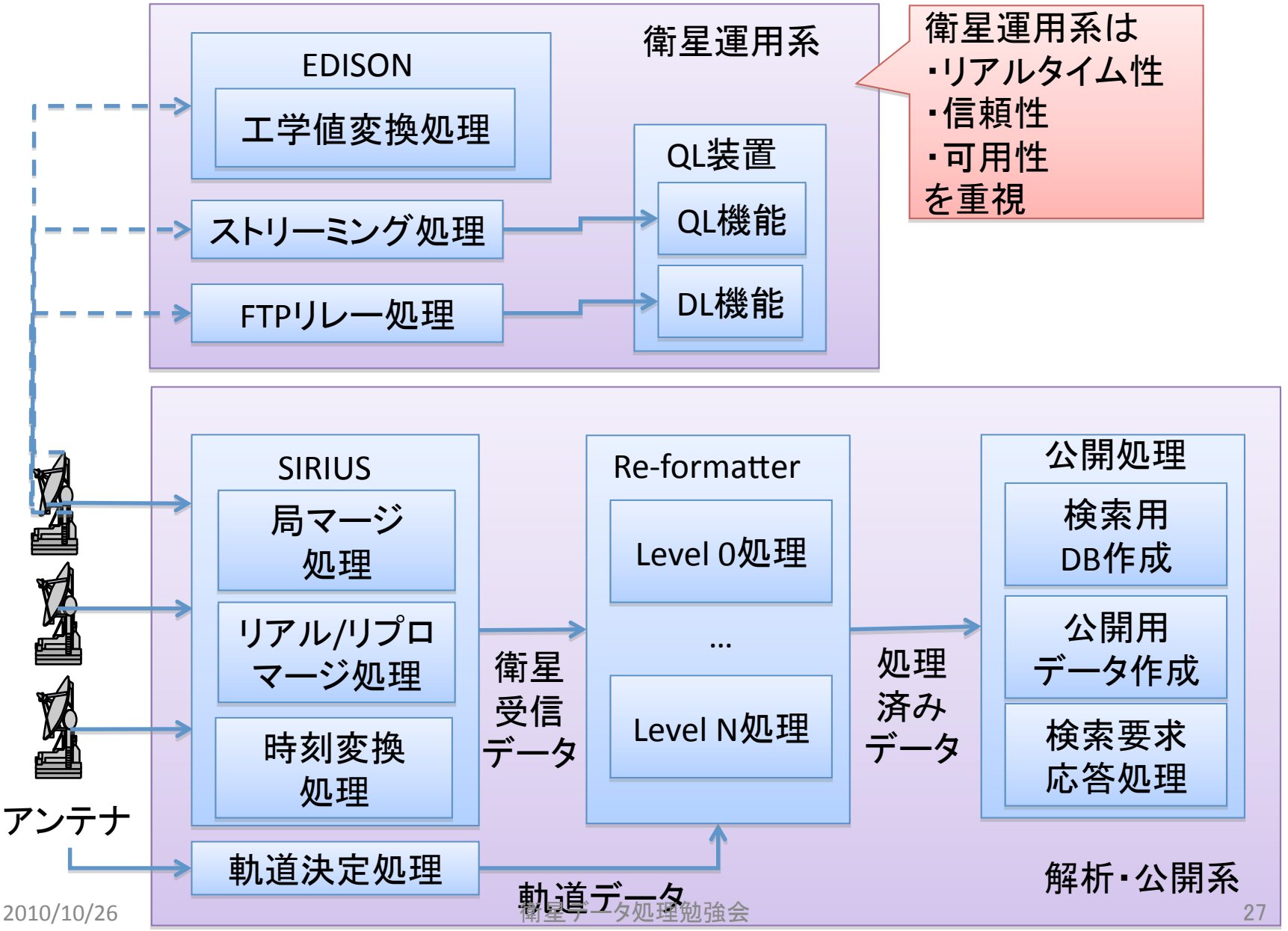
左の例では、2000年1月11日に
1:name:suzuki
1:age:25
1:hobby: baseball
を登録したが、2005年10月12日に
1:age:30
1:hobby:computer
へと更新した。
更新履歴は指定した世代まで
データベースは保持しており、
前の世代の値を取り出すことも可能。

実際にはタイムスタンプで数値が入っている

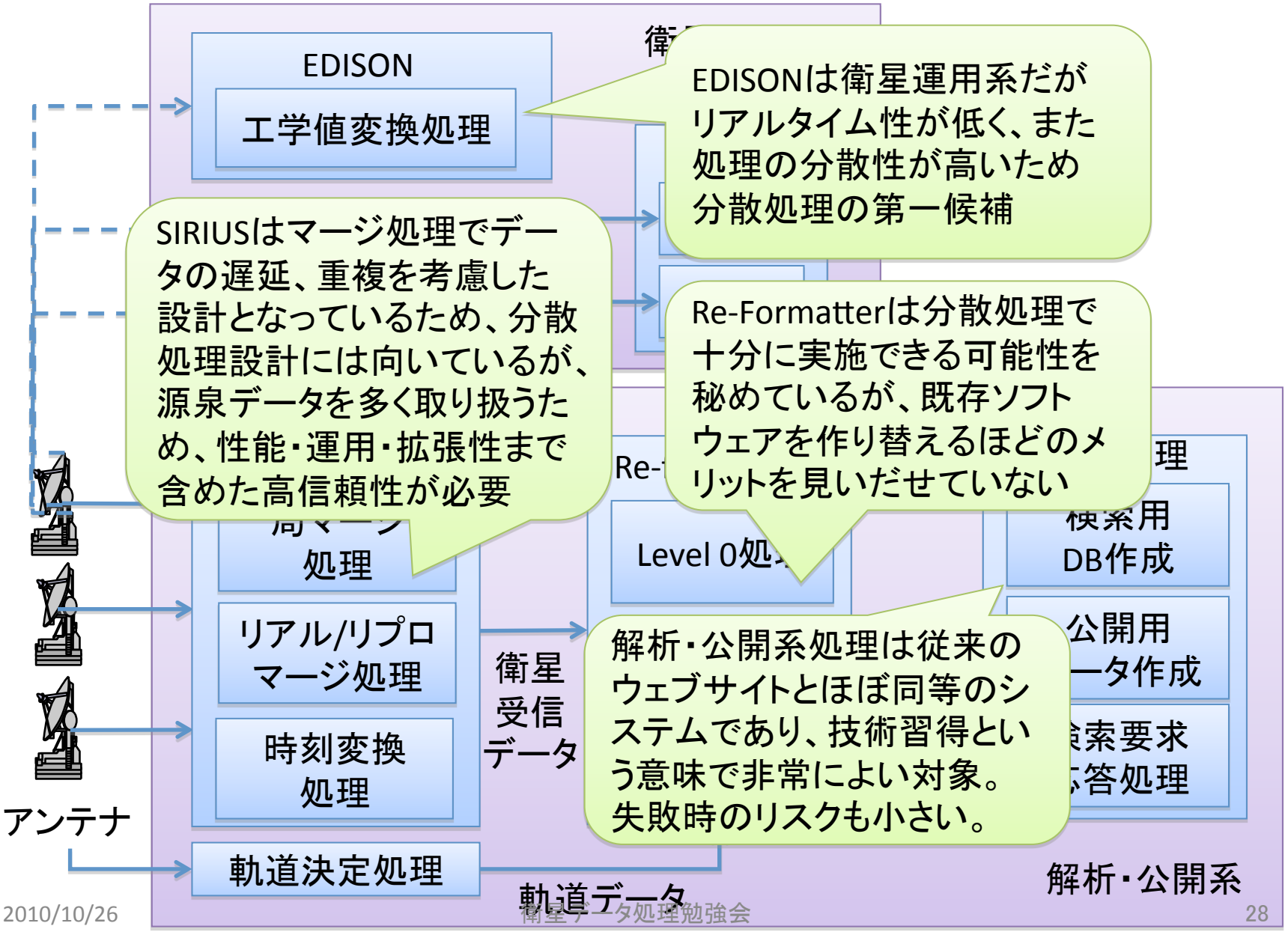
衛星データ処理とHadoop

Hadoopの使いどころは?

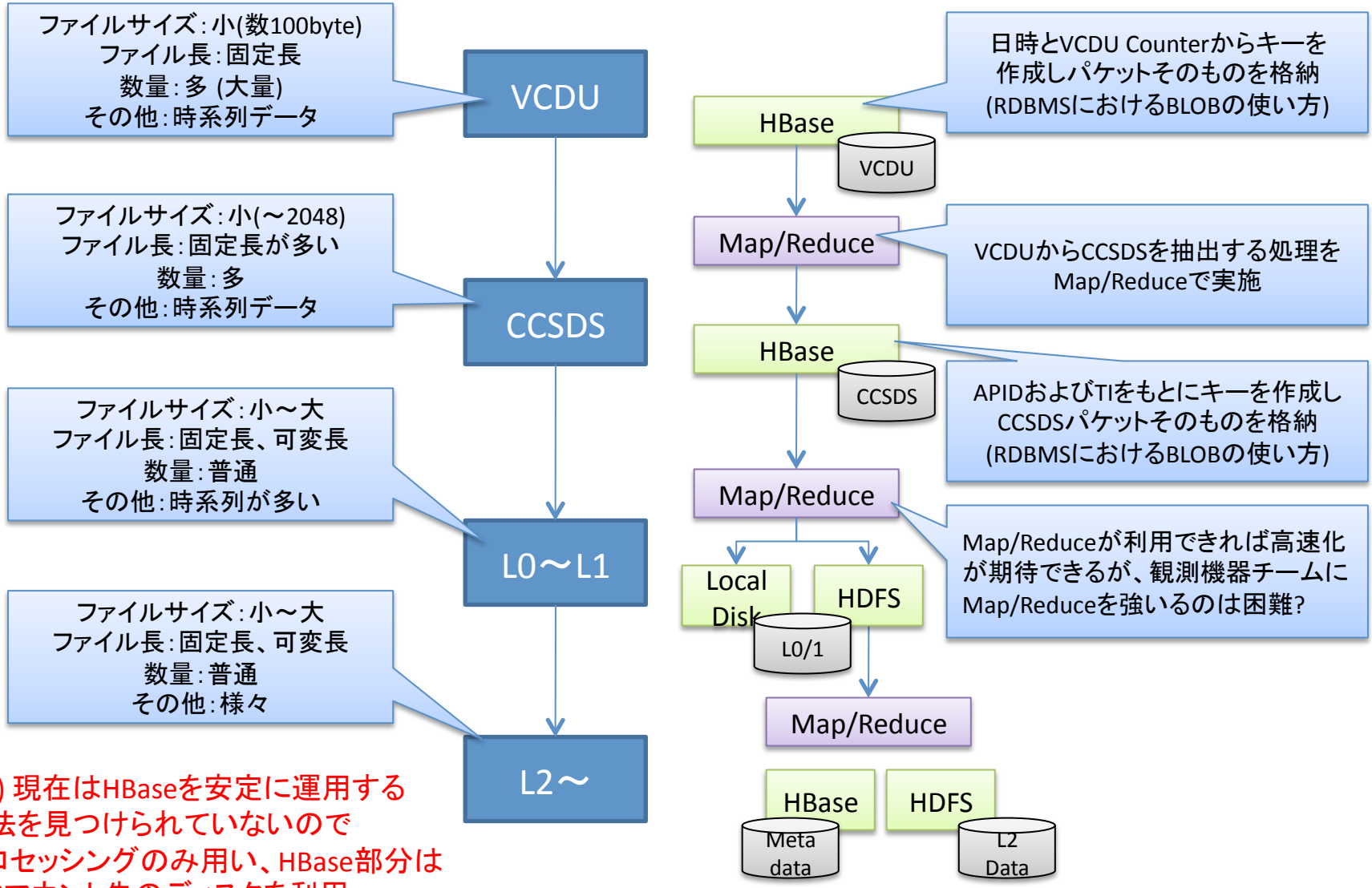
科学衛星運用系と解析・公開系処理



科学衛星運用系と解析・公開系処理



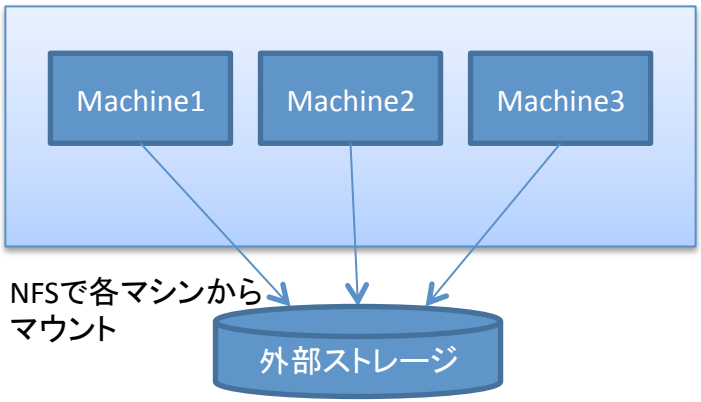
衛星データの特徴とテクノロジーの選択



(注) 現在はHBaseを安定に運用する
方法を見つけられていないので
プロセッシングのみ用い、HBase部分は
NFSマウント先のディスクを利用

システム構成と処理方式

(1) データは外部ストレージに置き、処理のみHadoopで分散



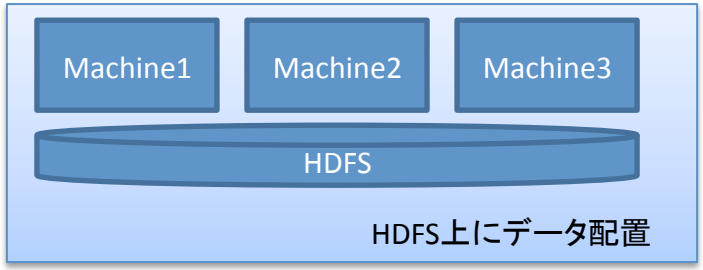
file:// + ファイルパスでアクセス

メリット
・他のシステムと連携しやすい
・HDFS上の制約に捕らわれない

デメリット
・台数が増えるとNFSがボトルネック(スケールアウトしにくい)
・処理するためにファイルリストを作る必要があるが、ファイル数が少ない場合にはファイルリストのサイズも小さく、結果として分散数が減少してしまう

処理方法案
・処理対象となるファイルリスト一覧を作成し引数で与える
・TextInputFormatを用いてファイルパスをvalueにセットする

(2) データをHDFSに置き、処理もHadoopで分散



メリット
・分散効率が最も高い
・スケーラビリティが良い(スケールアウトする)

デメリット
・HDFS上にデータを置くのに時間がかかる
・小さい大量のファイルをHDFS上に置きにくい
(Heapメモリの枯渇)

処理方法案
・ディレクトリを引数にしてディレクトリ内のファイルを一括処理

用途に応じて(1),(2),(1)+(2)の方式を使い分けることが現実的

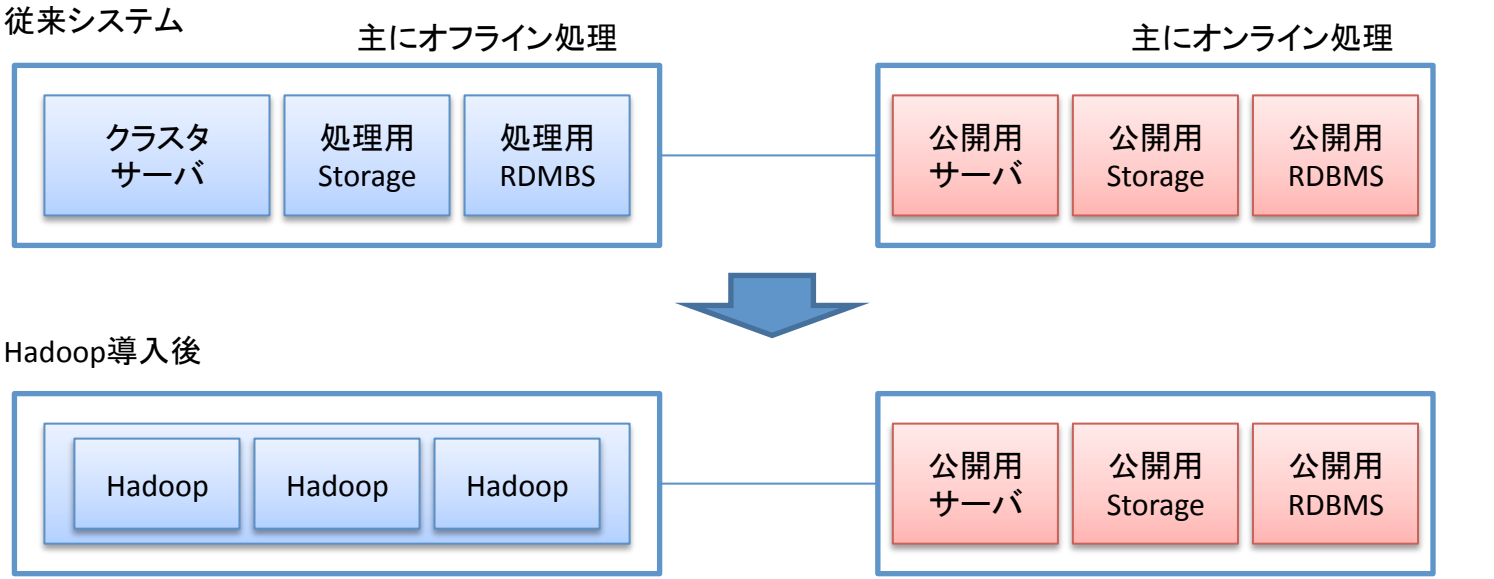
利用者と利用形態

- インフラ担当者 (SIRIUS,EDISON etc.)
  - C言語が主体 ⇒ 標準入出力を用いてHadoop Streaming
  - 将来的にはPigやJavaなどを用いて記述することは可能
- 低・高次処理担当者 (プロダクト作成)
  - C言語が主体 ⇒ Hadoop Streaming, Thrift API
  - この部分を研究者が担当することが多いためHadoop Streamingを推奨
- 公開系システム開発者
  - Java/php/perlが主体 ⇒ Pig/Java/Thrift API(主にプログラマーなので幅広い利用を期待)
- 研究者(数値計算・パラメータサーチ)
  - C言語やFORTRAN(特にスパコン関係者)が主体 ⇒ Hadoop Streaming (推奨), Thrift API
  - 有料ライセンスのソフトウェア利用(IDL, MatLab, etc.) ⇒ 単一で起動可能なStaticバイナリが作れなければHadoopの利用は諦める
  - 新しい言語・スクリプトの教育は困難

お金の話

Hadoopは本当に安いのか?

システム全体から見た位置付け



公開データを作る部分は十二分に利用可能

公開に関する部分はおそらく代替できない
・HDFSのスループットが悪い
・カラム型データベースがRDBMSと等価でない

従来システムをHadoopで代替可能な部分が大きいほど導入メリットがあるため、導入する前これを見極めることが重要
・公開部分が少ない (RDBMSがない, WebDAVや動画配信などファイル転送がすくない)
・ コンビニのPOSシステムやFirewallログの処理など、膨大なログ解析処理には向いている

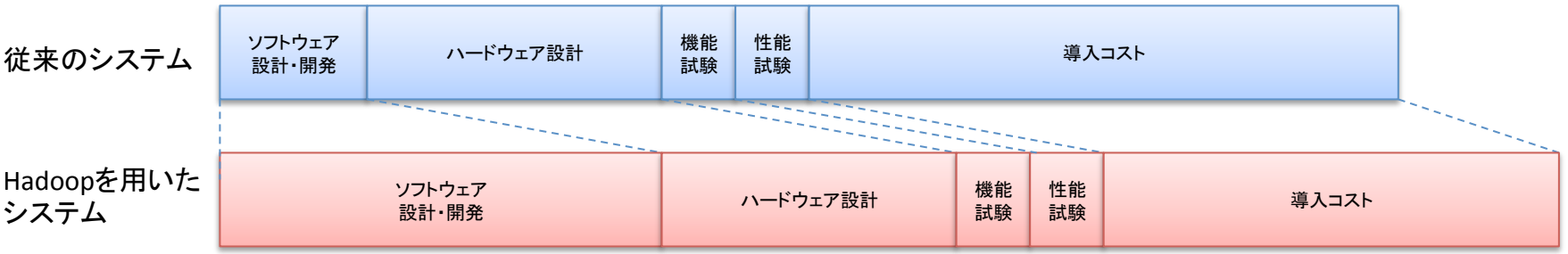
Hadoopを導入すると何が変わるか?

設計・開発から運用まで必要なコストの内訳が変わる (システム規模・内容によって変化するためあくまで参考)

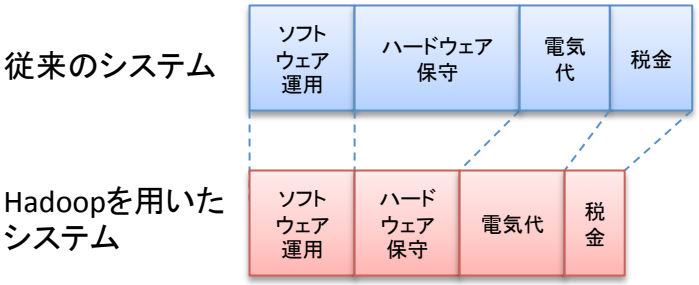
カテゴリ	サブカテゴリ	Hadoop導入のインパクト	負担
設計・開発コスト	ソフトウェア設計	HDFS,Map/Reduce,hBaseは設計が画一化されていないため大変。特に処理アルゴリズムをMap/Reduceに落としにくい場合は工夫が必要。将来的にはパターン化されるため現在のソフトウェア設計の負担と同程度まで落ちる可能性あり。	増加
	ハードウェア設計	ハードウェア台数が増加するため、電力・空調や設置スペース・耐震、ネットワーク構成、電源配線などを考慮するのが主な作業。ハードウェア1台あたりの処理能力に関しては少ない台数で実験して実測する。スペックが足りない場合はハードウェアを追加すれば対応可能であるため考慮しない。	設計項目が変化
試験コスト	機能試験	変化なし	変化なし
	性能試験	性能が足りない場合はハードウェアを追加すれば良いため性能確認が主作業。	試験項目が変化
導入コスト	ハードウェア購入	特殊ハードウェアではなく普通に購入できる汎用ハードウェアで構成されるため低減	減少
運用コスト	ソフトウェア運用	変化なしor大幅増(ライセンス費用が発生する場合は莫大な予算が必要or使えない)	増加?
	ハードウェア保守	汎用ハードウェアで構成されるため低減。ハードウェアが壊れた場合は追加購入すればOK(ただし今のところHDFSのNameNodeには高可用性必須)	減少
	性能追加	電力の見直し・設置スペース等のみで可能。高性能ハードウェアを購入し、ソフトウェアを再インストールすることを考えると低減。	減少
	設置スペース	ハードウェア台数が増加しているため増大。	増加
	電力	ハードウェア台数が増加しているため必要な電力は増加。	増加
	固定資産税	汎用ハードウェアで構成されるため低減する可能性大(消耗品扱い)。	減少
	廃棄	ハードウェアが老朽化した際には定期的大量に廃棄が発生する(リースという手も)。	増大
	リプレース	最新の機能を追加しなければ不要(随時最新ハードウェアで構成)	減少

コスト内訳・総額の変化(概算)

何もない状態から導入を考えた場合に必要な初期導入コスト



運用に必要なコスト



コストから見たHadoop導入のメリット・デメリット

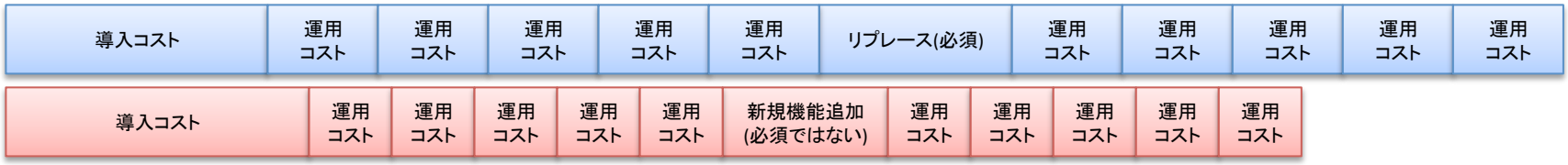
メリット

・ハードウェア保守費用の代わりに新規ハードウェアを購入
→ 性能が最新のスペックに追随しやすい
・長い目で見た場合運用コストは安い

デメリット

・初期導入コストが高い
・電気代が高い
・台数が多いのでいろいろ大変(監視や停電対応 etc.)
・設置スペース(= MONEY)が必要

長期計画で比較 (5年でリプレースを迎えるシステムの場合)



全体のまとめ

Summary

- Hadoopは分散ストレージ(HDFS)・分散処理(Map/Reduce)・分散データベース(HBase)を提供する
- 分散ストレージだけでなく分散処理も併用しないとコスト的に元が取れない
- オフライン処理でかつ分散性の高いSIRIUS, EDISON, Reformatter処理はHadoopに向いており、潜在的な可能性は高い
- 分散処理環境を意識することなく、行いたい処理に集中してプログラミングを記述できる
- スパコンの代替としてパラメータサーチ等の分散性の高い処理は代替可能だが、研究者に対してHadoop Streaming以外の利用方法を教育するのはコストが高く非現実的
- Hadoopは既存システムの全てを代替するものではない
- 基幹システムとして導入するには「HDFS Namenodeの高可用性の実現」や「Datanodeの監視・追加運用」等、運用ノウハウが必要で専用の運用部隊が必須
- 大規模でないと導入・維持に掛かる運用コストが高く、また技術が新しいため、現時点では精度の高いコスト見積もりが困難（余裕を持ったハードウェア数が必要)

科学衛星と言わず全JAXA規模で実施した方が幸せ

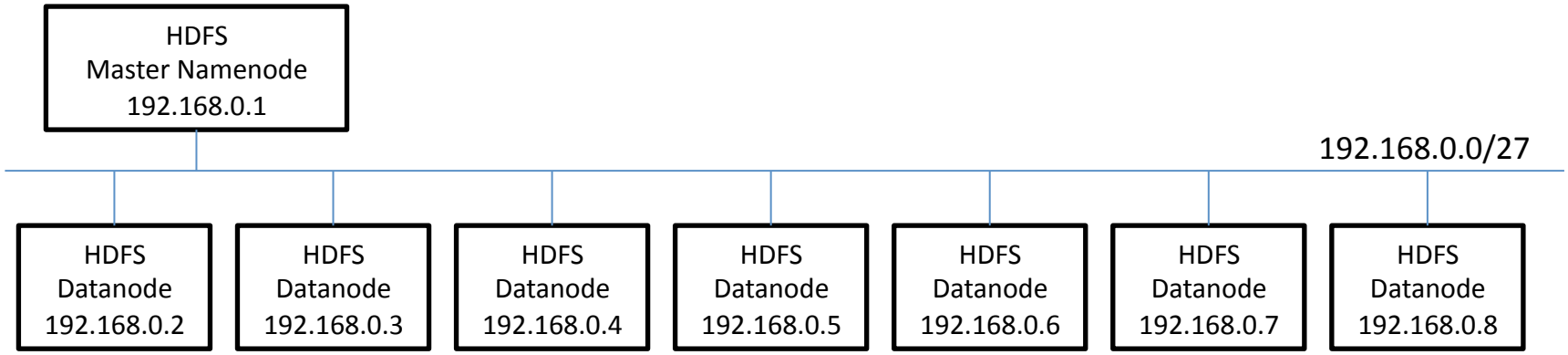
Appendix A

HDFSベンチマーク

HDFS Test Configuration

Machine x 8
Product: DELL PowerEdge R300
CPU: Core 2 Duo E6305 (1.86GHz,2MB L2 Cache, 1066MHz FSB)
Memory: 2GB (2x1GB 1R, DDR2/667MHz SDRAMM DIMM ECC)
HDD: 1TB 3.5 inch SAS HDD (7200rpm)
NIC: Broadcom BCM95722 PCI-E (Gigabit Ether)
Power Supply Unit: 400W
OS: CentOS 5.4 (x86_64)

L2 Switch
Product: Cisco Catalyst 2960G-24TC
Switch Configurations: 24 Ethernet 10/100/1000 ports
Power Rating: 0.075kVA (about 75W)



Local Disk Benchmark
/sbin/hdparm -Tt /dev/sda
/dev/sda:
Timing cached reads: 3716 MB in 2.00 seconds = 1857.65 MB/sec
Timing buffered disk reads: 318 MB in 3.01 seconds = 105.50 MB/sec

Network Benchmark
[dell02 ~]$ netperf -H 192.168.0.1
TCP STREAM TEST from 0.0.0.0 (0.0.0.0) port 0 AF_INET to 192.168.0.1 (192.168.0.1) port 0 AF_INET
Recv Send Send
Socket Socket Message Elapsed
Size Size Size Time Throughput
bytes bytes bytes secs. 10^6bits/sec
87380 16384 16384 10.03 941.33

HDFS Benchmark

HDFS Write Benchmark (Just after Format)
$ hadoop jar /usr/lib/hadoop/hadoop-0.20.1+169.56-test.jar TestDFSIO -write -nrFiles 10 -fileSize 1000

----- TestDFSIO ----- : write
Date & time: Wed Feb 17 21:32:46 JST 2010
Number of files: 10
Total MBytes processed: 10000
Throughput mb/sec: 11.457129141322543
Average IO rate mb/sec: 11.606027603149414
IO rate std deviation: 1.3865530212700998
Test exec time sec: 123.429

HDFS Read Benchmark (Just after Format)
$ hadoop jar /usr/lib/hadoop/hadoop-0.20.1+169.56-test.jar TestDFSIO -read -nrFiles 10 -fileSize 1000

----- TestDFSIO ----- : read
Date & time: Wed Feb 17 21:34:51 JST 2010
Number of files: 10
Total MBytes processed: 10000
Throughput mb/sec: 35.64604899192973
Average IO rate mb/sec: 98.31224822998047
IO rate std deviation: 194.86022307147937
Test exec time sec: 67.875

Appendix B

Hadoopトラブルアラカルト

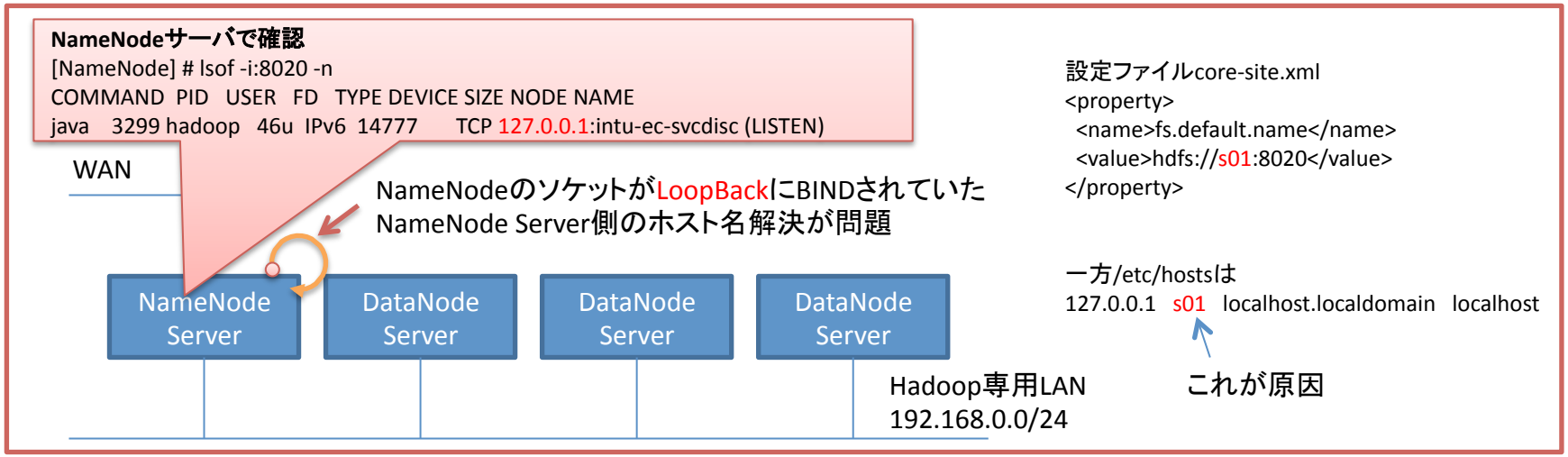
セットアップでハマった事

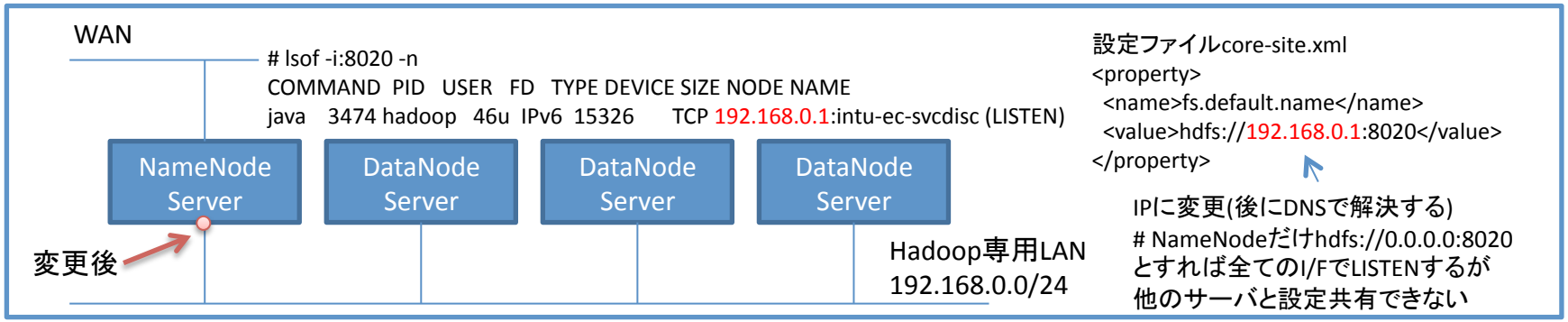
DataNodeサービスがNameNodeに接続できない

Hadoop専用LAN内のホストからNameNode Serverのport 8020に接続できない→/etc/hostsが問題だった

DataNodeのログ

```
2010-02-12 00:07:20,076 INFO org.apache.hadoop.ipc.RPC: Server at s01/192.168.0.1:8020 not available yet, Zzzzz...
2010-02-12 00:07:22,081 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: s01/192.168.0.1:8020. Already tried 0 time(s).
2010-02-12 00:07:23,084 INFO org.apache.hadoop.ipc.Client: Retrying connect to server: s01/192.168.0.1:8020. Already tried 1 time(s).
```





セットアップでハマった事

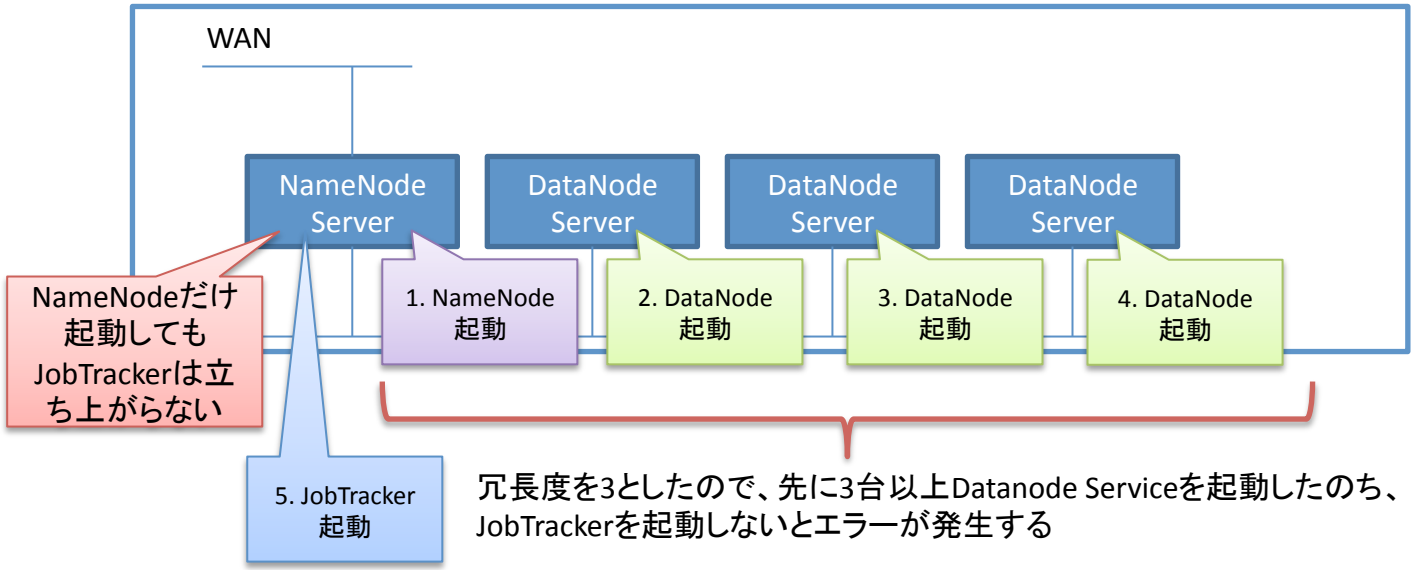
JobTrackerサービスが立ち上がらない(1)

以下のようなエラーが発生

INFO org.apache.hadoop.ipc.Server: IPC Server handler 6 on 8020, call addBlock(/tmp/hadoop-hadoop/mapred/system/jobtracker.info, DFSClient_-43795167, null) from 192.168.0.1:58235: error: java.io.IOException: File /tmp/hadoop-hadoop/mapred/system/jobtracker.info could only be replicated to 0 nodes, instead of 1
java.io.IOException: File /tmp/hadoop-hadoop/mapred/system/jobtracker.info could only be replicated to 0 nodes, instead of 1

/etc/hadoop/conf/hdfs-site.xmlの設定

```
<property>
  <name>dfs.replication</name>
  <value>3</value>
</property>
```

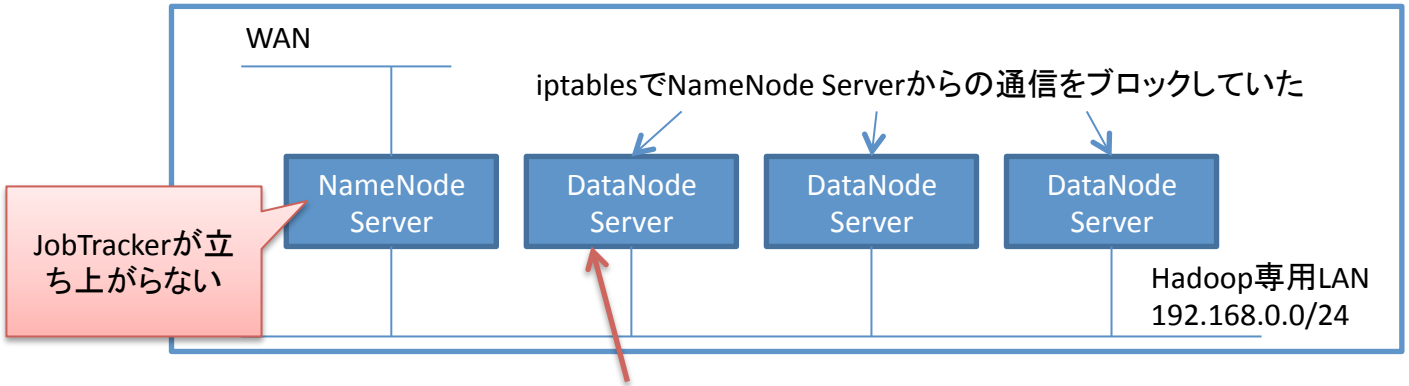


冗長度を3としたので、先に3台以上Datanode Serviceを起動したのち、JobTrackerを起動しないとエラーが発生する

セットアップでハマった事

JobTrackerサービスが立ち上がらない(2)

Datanodeサービスを立ち上げる際にiptablesがブロックしていた



待ち受けポートは以下

```
[root@dell02 ~]# lsof -n -itcp  | grep java | grep LISTEN
java    16312 hadoop   42u  IPv6  68978      TCP *:50688 (LISTEN)
java    16312 hadoop   55u  IPv6  69040      TCP *:50010 (LISTEN) ← dfs.datanode.address
java    16312 hadoop   56u  IPv6  69042      TCP *:50075 (LISTEN) ← dfs.datanode.http.address
java    16312 hadoop   61u  IPv6  69057      TCP *:54203 (LISTEN)
java    16312 hadoop   62u  IPv6  69048      TCP *:50020 (LISTEN) ← dfs.datanode.ipc.address
java    16492 hadoop   43u  IPv6  69439      TCP *:50060 (LISTEN) ←  mapred.task.tracker.http.address
java    16492 hadoop   50u  IPv6  69455      TCP 127.0.0.1:50968 (LISTEN)
```

とりあえずの処置例

全DataNode Serverでiptablesを停止

```
# /etc/init.d/iptables stop
```

再起動時にも起動しないように設定

```
# chkconfig   iptables   off
```

iptablesの設定を変更する例

/etc/sysconfig/iptablesに下記を追加 (CentOS 5.4の場合)

```
-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 50010 -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 50075 -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 50020 -j ACCEPT
-A RH-Firewall-1-INPUT -m state --state NEW -m tcp -p tcp --dport 50060 -j ACCEPT
```

その後再起動

```
# /etc/init.d/iptables restart
```

セットアップでハマった事
1台から複数台に変更した際のトラブル

現象：HBaseがクラスタモードで動かない
問題点：regionserverが起動しない → regionserverのホスト名解決が原因
解決方法：各サーバの/etc/hostsからregionserverのホスト名を削除後ZooKeeper再起動

現象：Rsyncで複製した設定が反映されない
問題点：yumでインストール, 設定の複製後alternativeを実行していなかった
解決方法：次のコマンドを実行
alternatives --install /etc/hbase-0.20/conf hbase-0.20-conf /etc/hbase-0.20/conf.my_cluster 50

現象：1台のとき動いたMap/Reduceが動かない

問題点1：エラー発生 java.lang.NoClassDefFoundError: org/apache/zookeeper/Watcher
解決方法：
/etc/hadoop/conf/hadoop-env.shのHADOOP_CLASSPATHにzookeeper-3.2.2.jarを追加

問題点2：エラー発生 java.lang.RuntimeException: java.lang.ClassNotFoundException
⇒jarで固めていないor 固めたjarファイルの構造がおかしい
解決方法：
必要なファイル・クラスをjarで固めてhadoop jar 固めたjarファイルで実行する

開発でハマった事
NoClassDefFoundError HBaseConfiguration

現象:
$HADOOP_HOME/conf/hadoop-env.shのHADOOP_CLASSPATHに$HBASE_HOME/conf/hbase-0.20.3.jarを追加してもHBaseConfigurationが見つからないというエラーが発生する。

問題点:
HBASE_HOMEを.bashrcで設定し、hadoop-env.shの中で環境変数HBASE_HOMEを用いていたことが原因。
Map/Reduce動作時に$HBASE_HOMEが適切に展開されずに見つからないとエラーになった。

解決方法:
hadoop-env.shの上の方に
export HBASE_HOME=…
を追加した。

開発でハマった事
Hadoopに流したJOBが止まらない

現象:
HBaseにデータを登録するプログラムを実行し、Ctrl-Cでプログラムを止めてもレコードが次々と登録される

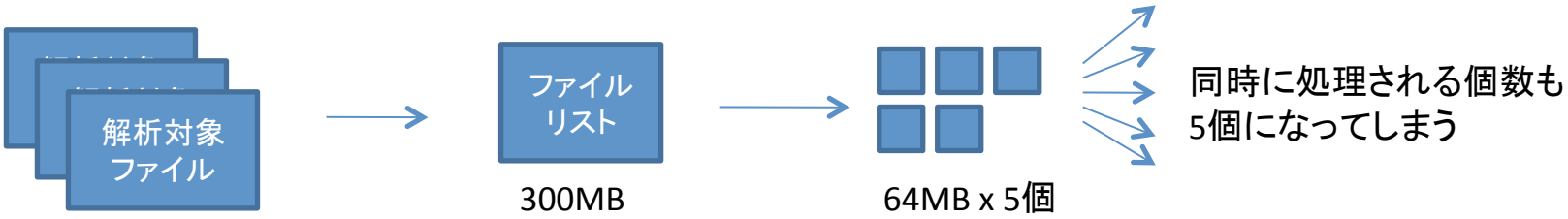
原因:
Jobを正しくkillできていなかった
・Ctrl-Cでコンソール上のForegroundタスクは止めていたがJobがそのまま残っておりレコードが次々と作成される

解決方法:
hadoop job -list
でジョブを表示した後、killしたいjobに対して
hadoop job -kill [JobIdの文字列]

開発でハマった事
処理が分散されない (1)

現象
Map処理が分散されない

原因
・入力となるファイルがHDFS上で十分に分散されていないため、処理が分散されていなかった



NAS上の解析対象ファイルをHDFS上にファイルリストとして作成。ファイルサイズは約300MB。
HDFSのブロックサイズは標準で64MBなのでファイルリストは5個のブロックで格納されている。
この場合5台のマシンで処理が進むため、ハードウェア台数を増やしても、実質的には5台で
しか処理が進まない。

対策
・解析対象ファイルをHDFS上に置くことが可能であれば置き、ディレクトリを指定して処理を行うようにする

開発でハマった事
HBaseが不安定

現象: 途中でHBaseのRegion Serverが落ちる

原因: とりあえず不明
データ挿入中にhbase shellでシェルを起動していろいろすると落ちやすい。特にcount ‘テーブル名’で進行状況の確認する際、レコード数が膨大だと高確率でRegion Serverが落ちる。

解決:
・挿入と参照はタイミングを分ける
・HBaseを使うことを止める (Cassandraがいいかも？)

開発でハマった事
HBaseがボトルネック

現象
Map/Reduceで分散されていて処理速度が速くなっているはずなのに単体で動かした方が速い

原因
HBaseがボトルネックで詰まり結果処理が遅延していた。複数台数のマシンから1台のHBase masterにアクセスすることにより、高負荷状態が続いたことが原因。さらにHBaseへの書き込みを高速化するために複数レコードをメモリ上に置いて一度に書き込んだ場合は処理が進まないで止まってしまう。

解決
・HBaseへの書き込みはMap/Reduceを使わず専用のソフトウェアを用意して単体で動作させる。
・Map/Reduceでやるなら遅くても我慢して1レコード1書き込みを徹底する。

```
$ hadoop jar hbinsert.jar hbinsert.Insert /somewhere/dir1/
```
を
```
$ export HBASE_CLASSPATH=hbinsert.jar
$ hbase hbinsert.Insert hdfs://master:8020/somewhere/dir1
```
に変更して速くなれば、HBaseがボトルネックになっている可能性が高い。後者はMap/Reduceを経由させていないため、ローカルファイルにも自由にアクセス可能。

開発でハマった事
WholeInputFormatを使った場合のvalue.getBytes()で得られるバイト数はファイルサイズとは違う

O'reillyに掲載されているWholeFileInputFormatおよびWholeFileRecordReaderをmap関数から呼び出した場合

```
void map(NullWritable key, ByteWritable value, …) {
  String filename = conf.get("map.input.file");
  byte[] data = value.getBytes();
   long  filesize = data.length; ←この行は間違い
}
```

のように実施するが、data.lengthがファイルサイズとは異なるため、これをファイルサイズと信じて使うと思わぬ不具合が生じる。

```
void map(NullWritable key, ByteWritable value, …) {
  String filename = conf.get("map.input.file");
  byte[] data = value.getBytes();
  long  filesize = conf.getLong("map.input.length",0); ← ファイルサイズはconfから取得する。
}
```

Appendix C

2つのHadoop

オリジナル版とCloudera版
どちらをインストールすればよいか?

- オリジナル版
  - メリット
    - 最新版が利用可能
    - サービスの起動が楽 (start-all.shで全関連サービスが起動)
  - デメリット
    - NameNodeサーバから自由にログイン可能なアカウントを作成する必要あり
    - PC起動時に動作させたい場合、起動スクリプトを書く必要あり
    - 設定ファイルが空
- Cloudera版
  - メリット
    - レポジトリでインストール可能(CentOS, Ubuntu)
    - 自由にログイン可能なアカウントを作る必要がない(hadoopユーザを自動作成するがパスワード付き)
    - 起動スクリプトが付いているためサービス化が楽(PC起動時にすぐ開始)
    - ブラウザ上で管理可能な「Clouderaデスクトップ」をインストールしやすい
    - HBase, ZooKeeper, Pigなどもレポジトリでインストールできる
  - デメリット
    - 若干バージョンが古い
    - PC一台一台サービスを起動・停止する必要がある(start-all.sh, stop-all.shのようなものがない)

本番運用→ サービスを頻繁に落としたり起動したりしない→Cloudera版
試験運用→サービスを頻繁に落としたり起動したりする→オリジナル版
とも言えるが、今のところおそらくどちらも変わらない(結局のところ好み)

Appendix D

Datanodeの増設

Data Nodeの追加 (オリジナル版)

買ってきたサーバをネットワークにつなげてすぐHadoop Datanodeとなれば良いのだが、事はそう簡単でない

1.OSのインストール

2. SUNのJavaをインストール

3.sshdのインストール
yum install sshd
chkconfig –level 2345 sshd on

4.Hadoopが通信するためのユーザを作成
adduser hadooper
passwd hadooper

5.Name Nodeサーバから自動ログインできるように設定
$ mkdir .ssh
$ chmod 700 .ssh

NameNodeサーバ側で
$ scp ~hadooper/.ssh/id_rsa.pub hadooper@(サーバip):/home/hadooper/.ssh/authrozied_keys
$ ssh hadooper@(サーバip)

必要に応じてiptablesの設定を変更する

Data Nodeの追加 (Cloudera版,CentOSの場合)

1.OSのインストール

2. SUN Javaをインストール

3.sshdのインストール
yum install sshd
chkconfig –level 2345 sshd on

4.Name Nodeサーバからレポジトリを複製
scp /etc/yum.repos.d/cloudera-*.repo root@(サーバip):/etc/yum.repos.d/

5. Hadoopのインストール
yum check-update
yum update
ここで念のため再起動
yum -y install hadoop-0.20
yum -y install hadoop-0.20-conf-pseudo-0.20.1+169.56-1 ← これがないと/var/lib/hadoop-0.20/cacheが作られない

NameNodeサーバ側で
$ rsync -e ssh -avz /etc/hadoop-0.20/conf.my_cluster/ root@(サーバip):/etc/hadoop-0.20/conf.my_cluster

DataNodeサーバ側で
alternatives --install /etc/hadoop-0.20/conf hadoop-0.20-conf /etc/hadoop-0.20/conf.my_cluster 50

必要に応じてiptablesの設定を変更する