大規模分散データ処理システム Hadoopの惑星科学データベースへの適用可能性検討

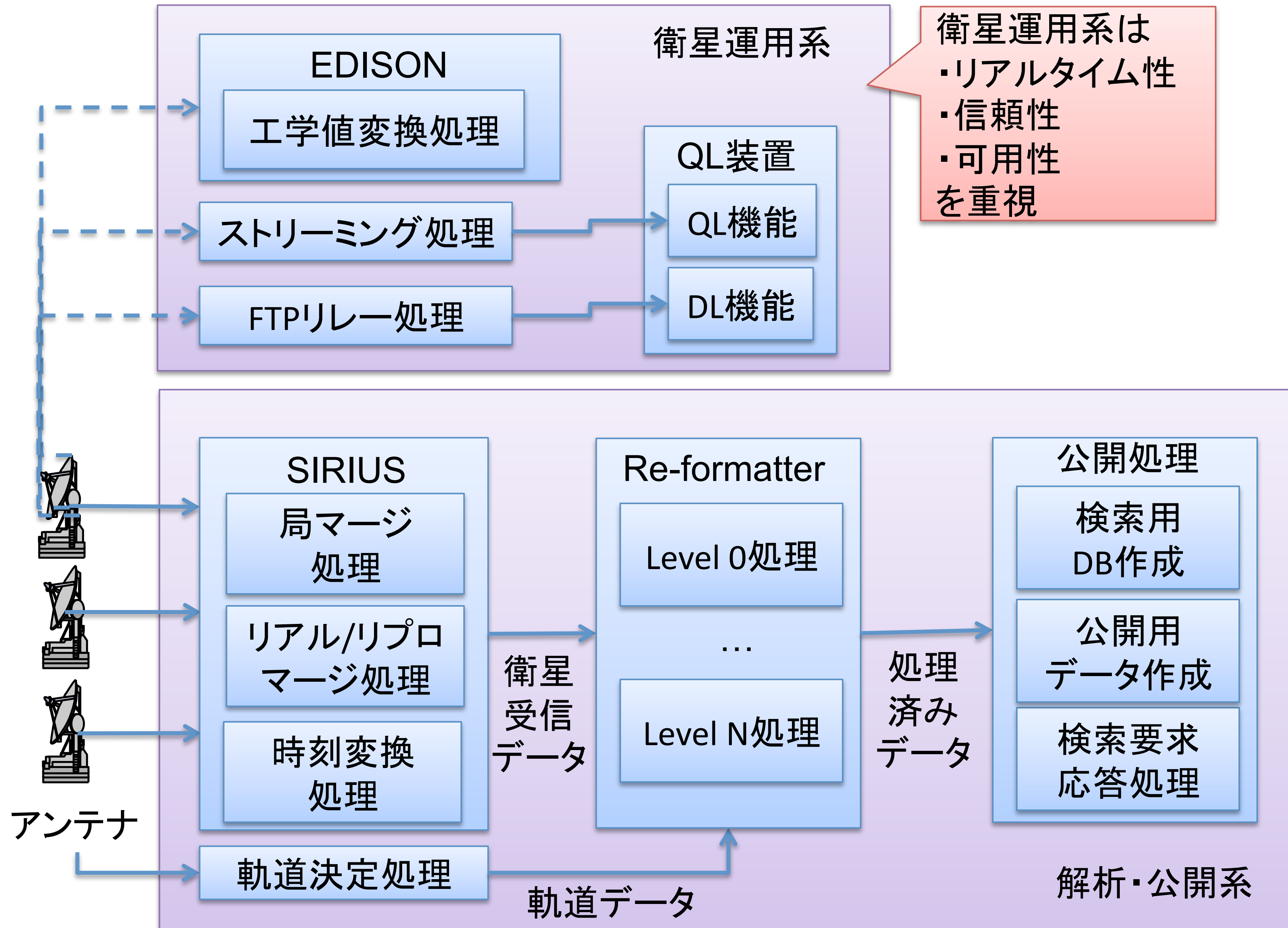
宇宙航空研究開発機構
JSPEC 研究開発室/
ISAS C-SODA
○山本 幸生

背景

- 月周回衛星かぐやのデータが約30TB、プロダクト数として1000万以上を予定
- 現在開発中のPlanetary Data Access Protocolはミッションを横断したデータベースを構築予定
- 増加し続けるデータに対して、RDBMSの検索性能はデータ量以上にリニア以上に劣化する事実
- 大量データを取り扱っているGoogleでメインに利用されているデータベースはカラム型データベース

惑星科学データベースを構築する上でどの程度利用可能なのか?

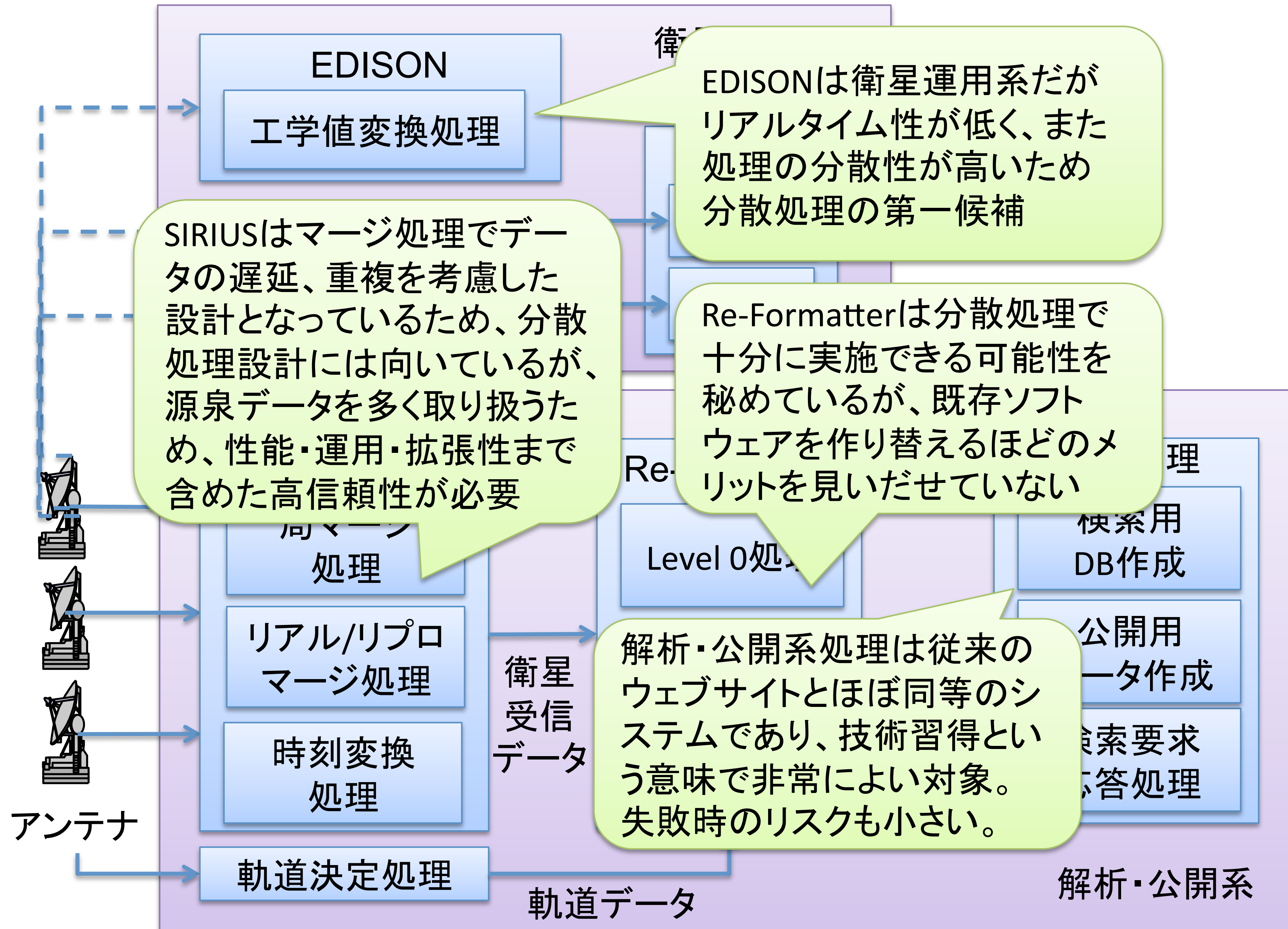
科学衛星運用系と解析・公開系処理



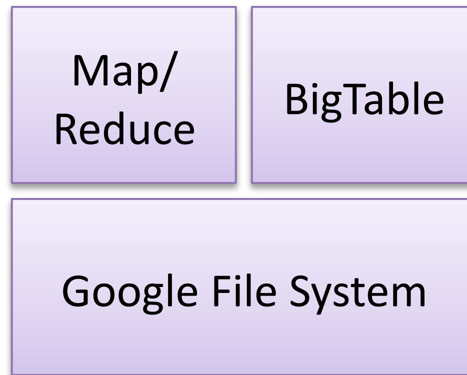
衛星データと分散処理の要求事項

- 衛星データに対する要求事項
  - 蓄積方法(データ処理容量)
  - 処理タイミング(リアルタイム性)
  - 信頼性
  - 可用性
- 分散処理に対する要求事項
  - 処理の分散性(処理の他処理への非依存性)
  - 非リアルタイム性(応答遅延)

科学衛星運用系と解析・公開系処理

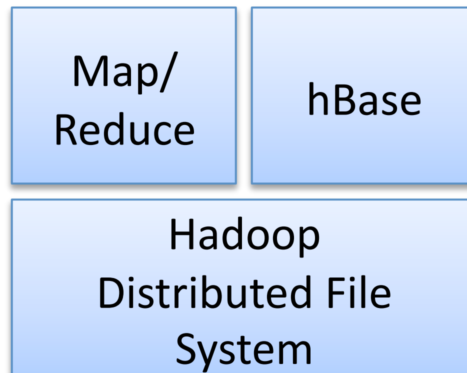


Hadoop概要



Google BigTableのオープンソースクローン。

分散ファイルシステムHDFS上に、MapReduceアルゴリズムとhBaseと呼ばれるカラム型データベースで構成。MapReduceとHDFSのみ、あるいはhBaseのみといった使い方も可能。

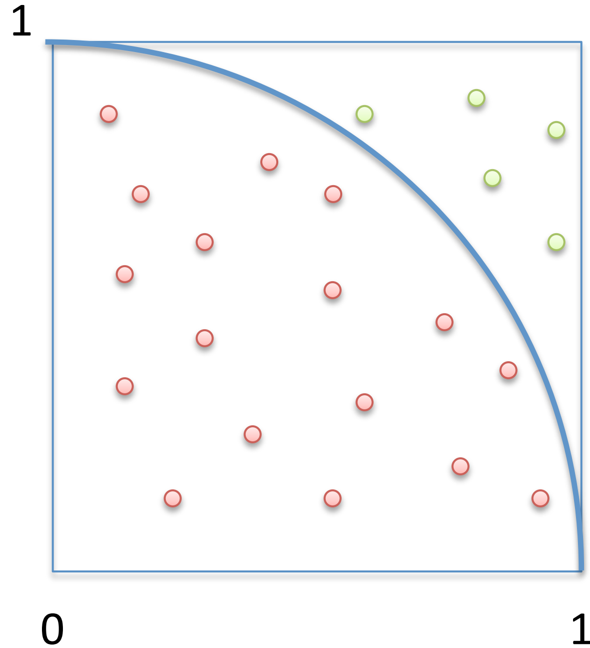


Google BigTableはGoogle App Engineで利用可能。PythonやJava、またその派生環境を利用してプログラム可能。

HadoopはJavaでプログラミング。他の言語を利用する場合、Hadoop Streamingを用いて標準入出力を介して実行可能。数100台のハードウェアを用意するのが困難な場合、Amazon EC2などを利用して環境を構築可能。

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>

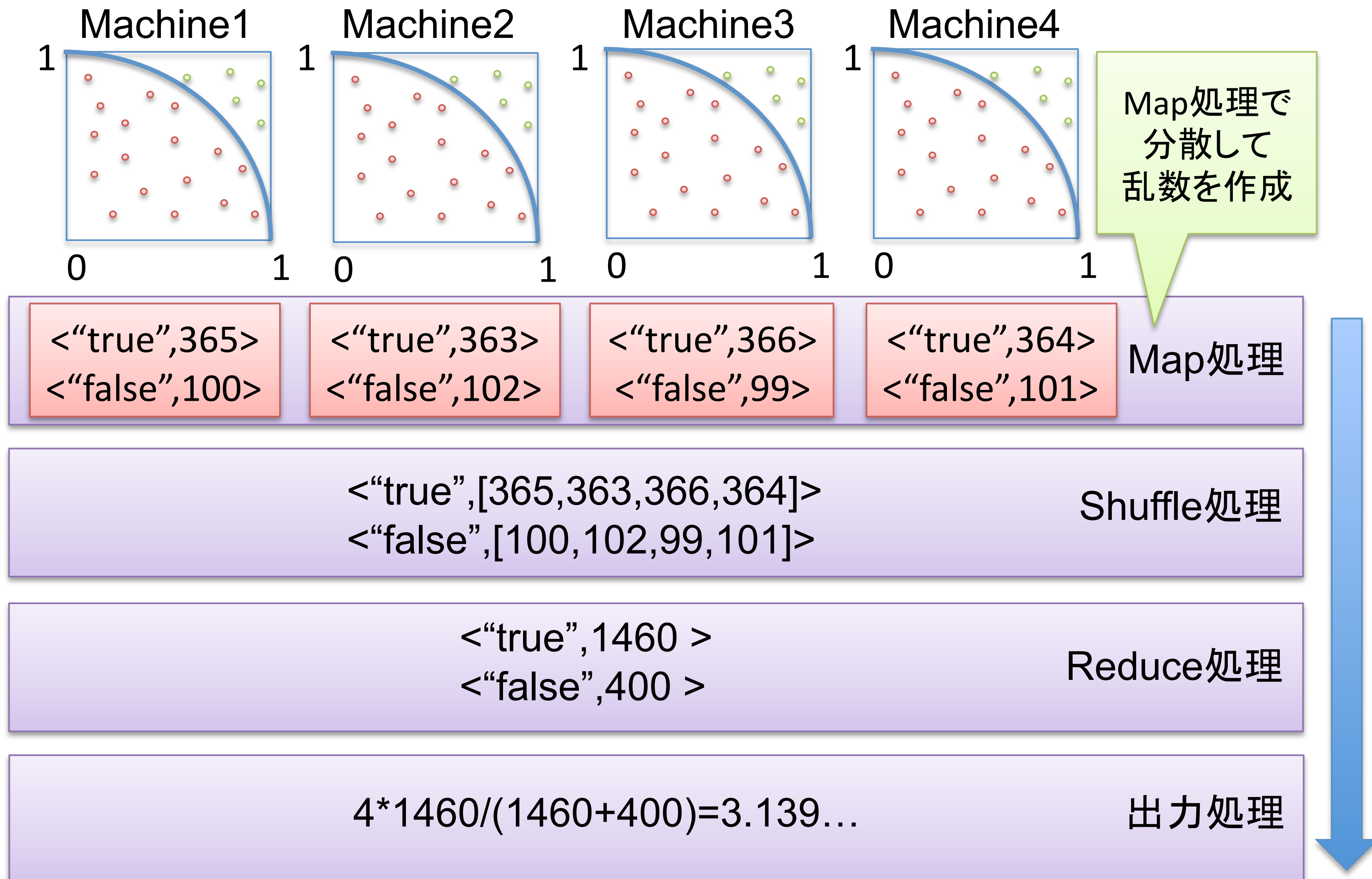
という<キー,値>のペアを作成する。このマップは分散された個数だけ生成されるので、Reduce処理によって集計する必要がある。

”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

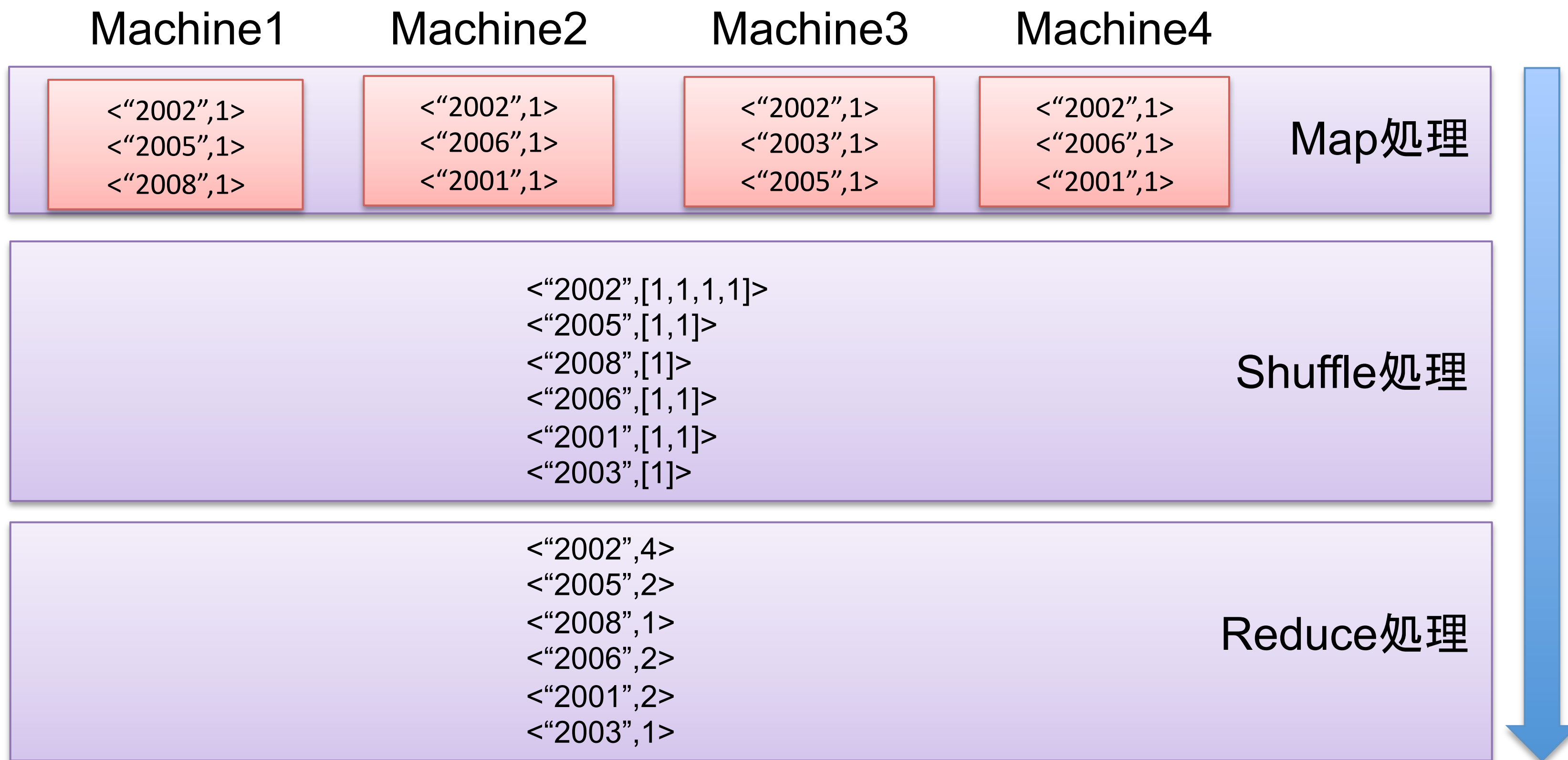
MapReduceによる円周率計算2



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]



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	Valu3	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。

通常のRDBMSと同様に扱えるが、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”,
}
```

↑ key ↑ value

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り、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で、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

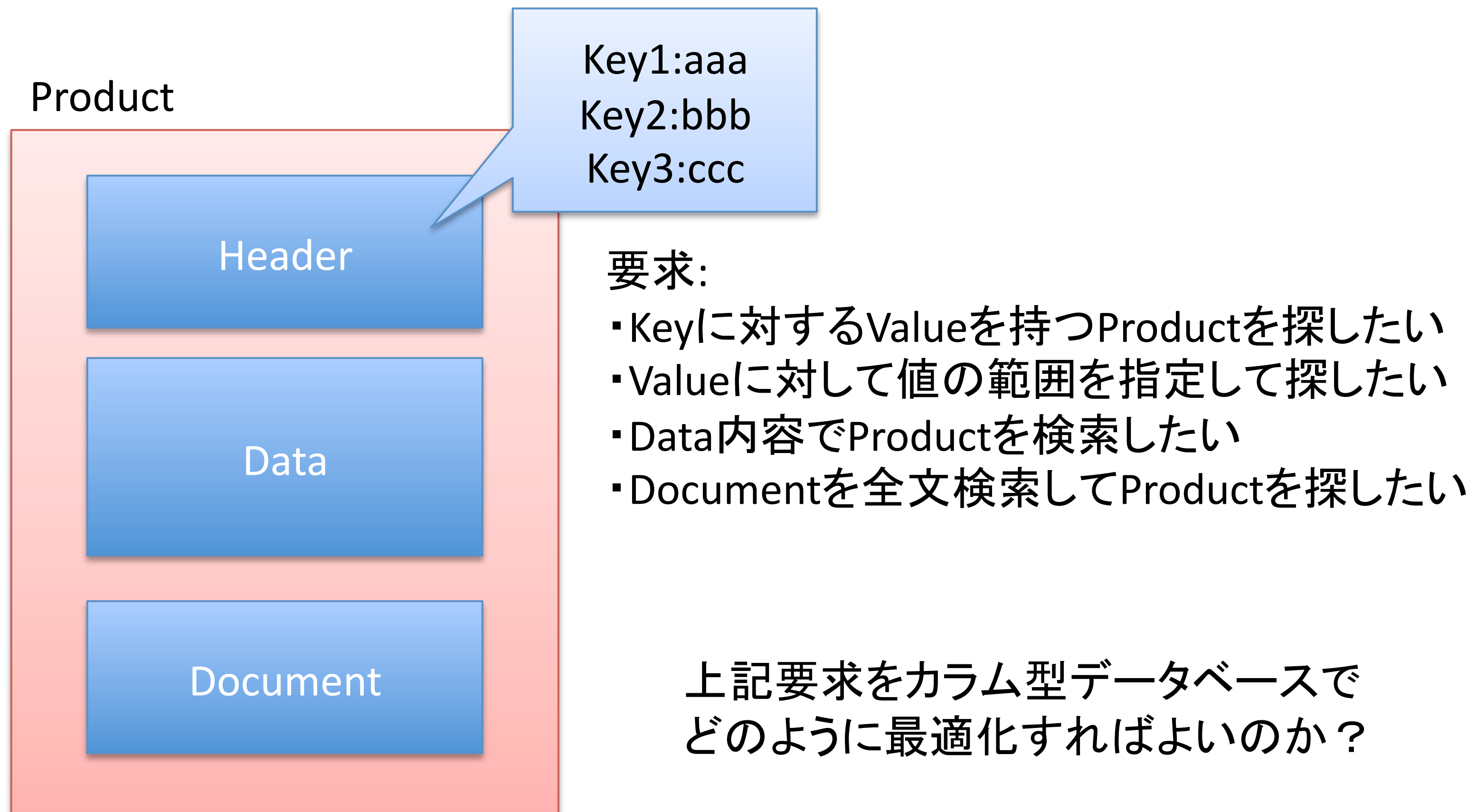
それぞれのCellの値はTimestampを記録しバージョンという概念がある

```
{
    "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UNIXタイムスタンプが利用される

惑星科学データの一般的な特徴



Hayabusa AMICAデータを用いたhBaseにおける設計

Table Name	Hayabusa
Row Key	ProductID
Column Family:Sub Column Family	header:<FITSキーワード>
Cell	<FITSキーワードに対する値>

Header Keyword	Value
PRODUCT_ID	ST_2447741779_b
LAST_VVALUE	1023
SUMDIF_1VALUE	DIFF
DAT_TYPEVALUE	SCIENCE
TEMP_0VALUE	-25.59

このようなヘッダは下記のようなカラムデータベース上のレコードとして登録するように設計。見た目はRDBMSとほぼ同じ。

hBase上では…

Row Key	header: LAST_VVALUE	header: SUMDIF_1VALUE	header: DAT_TYPEVALUE	header: TEMP_0VALUE
ST_2447741779_b	1023	DIFF	SCIENCE	-25.59

hBaseへの挿入プログラム例

```
public static void main(String[] args)  throws Exception {// 引数で渡されたFITSファイル名のチェック
    FitsFile fitsFile = new FitsFile(args[0]); // FITSファイル名からFITSオブジェクトを生成

    // HBase上にテーブルを生成（既に存在する場合は何もしない）
    HBaseConfiguration config = new HBaseConfiguration();
    HBaseAdmin admin = new HBaseAdmin(config);
    if (admin.tableExists("hayabusa") == false) {
        System.out.println("create table");
        HTableDescriptor tableDescripter = new HTableDescriptor("hayabusa".getBytes());
        tableDescripter.addFamily(new HColumnDescriptor("header:"));
        admin.createTable(tableDescripter);
    }
    Set set = fitsFile.fitsMap.keySet();
    Iterator iterator = set.iterator();
    Object object;
    HTable table = new HTable(config, "hayabusa");
    System.out.println("###### PRODUCT-ID = " + fitsFile.productId + " ######");
    System.out.println("Insert/Update Start=>\n");
    // FITSヘッダー部をテーブルに挿入（ProductID/Header:Key/Value）
    while(iterator.hasNext()){
        object = iterator.next();
        String key = "header:" + object;
        String value = (String)fitsFile.fitsMap.get(object);
        BatchUpdate productUpdate = new BatchUpdate(fitsFile.productId);
        productUpdate.put(key, value.getBytes());
        table.commit(productUpdate);
        System.out.println("      COLUMN:" + key + "\tVALUE:" + value);
    }
    System.out.println("\n<================= Insert/Update End");
}
```

テーブル「hayabusa」を作る準備

Column family「header」を作る準備
※ column familyはテーブル作成後、追加できない

テーブル「hayabusa」を指定

Row keyとして「productID」を設定

(key,value)ペアを(column family, cell)として登録

hBaseへの挿入プログラム実行例

```
$ java InsertFits ../testdata/20051010_31229/ST_2447741779_b.fits
###### PRODUCT-ID = ST_2447741779_b ######
Insert/Update Start=>

    COLUMN:header:LAST_VVALUE:1023
    COLUMN:header:SUMDIF_1VALUE:DIFF
    COLUMN:header:DAT_TYPEVALUE:SCIENCE
    COLUMN:header:TEMP_0VALUE:-25.59

＜…省略…＞

    COLUMN:header:EXP_0VALUE:1.31e-01
    COLUMN:header:FF_B_1VALUE:OFF
    COLUMN:header:SIMPLEVALUE:T
    COLUMN:header:FF_B_0VALUE:OFF

<================= Insert/Update End
```

hBaseへの検索プログラム例

```
public static void main(String[] args)  throws Exception {
    if(args.length != 2){
       System.out.println("java ScanFits [Key] [Value]");
       return;
    }

   HBaseConfiguration config = new HBaseConfiguration();

   HTable table = new HTable(config, "hayabusa");

   System.out.println("###### KEY[" + args[0] + "] = VALUE[" + args[1] + "] ######");
   System.out.println("Scan Start=>\n");
   for (RowResult row : table.getScanner(new String[] { "header:" + args[0]})) {
      for (Map.Entry<byte[], Cell> entry : row.entrySet()) {
         Cell cell = entry.getValue();
         String value = new String(cell.getValue());
         if(value.equals(args[1])){
            System.out.format("ProductID\t%s\n", new String(row.getRow()));
         }
      }
   }
   System.out.println("\n<================ Scan End");
}
```

Keyを指定(header:<key名>)

header:<key名>を持つマップを全スキャン

valueと一致した場合に出力

hBaseへの検索プログラム実行例

引数で指定されたKeyに対してValueと一致する値を持つProductのProductIDを検索

```
$ hadoop jar ScanFits.jar ScanFits TEMP_0 -25.59
09/06/25 17:59:03 INFO jvm.JvmMetrics: Initializing JVM Metrics with processName=JobTracker, sessionId=
…
09/06/25 17:59:04 INFO mapred.LocalJobRunner:
#####Reduce##### ProductID=ST_2447741779_b
09/06/25 17:59:04 INFO mapred.TaskRunner: Task:attempt_local_0001_r_000000_0 is done. And is in the process of
commiting
09/06/25 17:59:04 INFO mapred.LocalJobRunner: reduce > reduce
09/06/25 17:59:04 INFO mapred.TaskRunner: Task 'attempt_local_0001_r_000000_0' done.
09/06/25 17:59:05 INFO mapred.JobClient:  map 100% reduce 100%
09/06/25 17:59:05 INFO mapred.JobClient: Job complete: job_local_0001
09/06/25 17:59:05 INFO mapred.JobClient: Counters: 13
09/06/25 17:59:05 INFO mapred.JobClient:   FileSystemCounters
09/06/25 17:59:05 INFO mapred.JobClient:     FILE_BYTES_READ=28017
09/06/25 17:59:05 INFO mapred.JobClient:     FILE_BYTES_WRITTEN=56462
09/06/25 17:59:05 INFO mapred.JobClient:   Map-Reduce Framework
09/06/25 17:59:05 INFO mapred.JobClient:     Reduce input groups=1
09/06/25 17:59:05 INFO mapred.JobClient:     Combine output records=0
09/06/25 17:59:05 INFO mapred.JobClient:     Map input records=1
09/06/25 17:59:05 INFO mapred.JobClient:     Reduce shuffle bytes=0
09/06/25 17:59:05 INFO mapred.JobClient:     Reduce output records=0
09/06/25 17:59:05 INFO mapred.JobClient:     Spilled Records=2
09/06/25 17:59:05 INFO mapred.JobClient:     Map output bytes=73
09/06/25 17:59:05 INFO mapred.JobClient:     Map input bytes=0
09/06/25 17:59:05 INFO mapred.JobClient:     Combine input records=0
09/06/25 17:59:05 INFO mapred.JobClient:     Map output records=1
09/06/25 17:59:05 INFO mapred.JobClient:     Reduce input records=0
```

Hadoopを利用する上での問題点

- カラム型データベースの設計が困難
  - RDBMSにおける正規化という概念や標準的な設計手法というものが確立されていない
- コスト見積りが困難
  - RDBMSのようにデータベース構造、レコード数が決まれば大凡のコスト(設計・運用)が決まるほど、利用されていない
  - サポート業務を実施してくれる業者がいないため、サポート業務に要するコストが見積もれない
- 性能見積りが困難
  - スケールアウト方式のため、データ量の増加に伴う著しい性能の劣化が見られないと言われているが、設計に依存しないかや、何台増加させれば改善するのかなど、課題点が多い
  - 性能劣化時にRDBMSにおけるチューニングやSQLの改善といった定型的な手法が確立されていない
- 大規模でないとオーバースペック
  - 小規模データでの利用には向いておらず、小規模であっても相応のハードウェア台数を必要とし、少ない台数だとパフォーマンスが悪い
- WHERE区などのような条件設定
  - 値の一致ではなく、数値に対してある範囲からある範囲まで、といった細かい指定を行なった場合に、高速に動作するかどうか疑問(値をrow keyとする専用テーブルを作成?)

高信頼性用途にはまだまだ不安(将来的には十分可能性有り)

まとめ

- 衛星運用系およびデータ解析・公開系において、大規模分散型データベースが適用可能な範囲は多い。リアルタイム性を除くシステムのほとんどを代用可能である。
- カラム型データベースを利用してHayabusa Amicaの検索システムバックエンドを開発し、実用になることを確認した。ただし、カラム型データベースの設計として良いかどうかは検討する必要がある。
- 問題点の多くが「新しいシステムで枯れていない」ことに起因しているため、現段階で高い信頼性を要する箇所には適用困難である。しかしながら、将来的には十分利用検討する価値があるデータベースである。
- 今後の課題として性能調査(データベース設計、マシン性能、台数etc.)を実施する必要がある。
- hBaseを使わず、Map/ReduceアルゴリズムでGrep検索やそれ相当のプログラムを作成した方が良い可能性もある。