分散処理アプリ演習　第1回
Hadoopの概要

（株）NTTデータ

講義内容

1. Hadoopの概要
2. HDFSの概要
3. 演習：HDFSの操作
4. MapReduceフレームワークの概要
5. Hadoopクラスタの構成
6. Hadoopエコシステム
7. Hadoopの利用事例


SOFTWARE ENGINEERS EDUCATION PROGRAM FOR TOP
TOP
SOFTWARE
ENGINEERS

1. Hadoopの概要

従来の大規模データ計算の問題点

- 単一マシンで大規模なデータの計算処理をしようとすると、より高速のCPUや、より多くのメモリを搭載する必要がある
- CPU速度やメモリ容量が2倍になると、それらの価格は通常2倍より多くかかる
- 単一のマシンに搭載できるCPUの処理能力やメモリの容量には限界がある
- 大容量で信頼性の高い単一のストレージの価格は非常に高価である
- 単一のストレージに格納できるデータの容量には限界がある

従来の分散処理の問題点

- 従来の分散処理のプログラムは非常に複雑
  - ノード間の依存関係や同期を意識する必要がある
  - 障害を前提に設計する必要がある
- データは通常専用のストレージに格納され、計算時に計算ノードにコピーされる
  - データ量が大規模になると、計算ノードへのコピーにかかる時間が膨大となる

Hadoopのコンセプト

- コモディティ
  - 入手容易な一般的なサーバ(コモディティ・ハードウェア)を多数用いて、大規模データを分散して格納し、計算処理を分散して実行する
- スケーラビリティ
  - 単にサーバの台数を増やせば、それに比例してデータ容量や計算処理能力を増大させることができる
- 耐障害性
  - データは複数のコピーを持ち、それぞれのコピーは異なるサーバに格納される
  - 障害となった計算処理は他サーバで自動的に再実行される
  - 障害から復旧したサーバはシステム全体を再起動することなく再参加できる
- データローカリティ
  - できる限り、計算時にデータは移動させず、データが保存されたサーバ上で計算処理を行うことにより、データのネットワーク通信にかかる時間を削減する
- 開発容易性
  - アプリケーション開発者はノード間の関係を気にした複雑なプログラムを作成する必要はない

Hadoopの適用領域

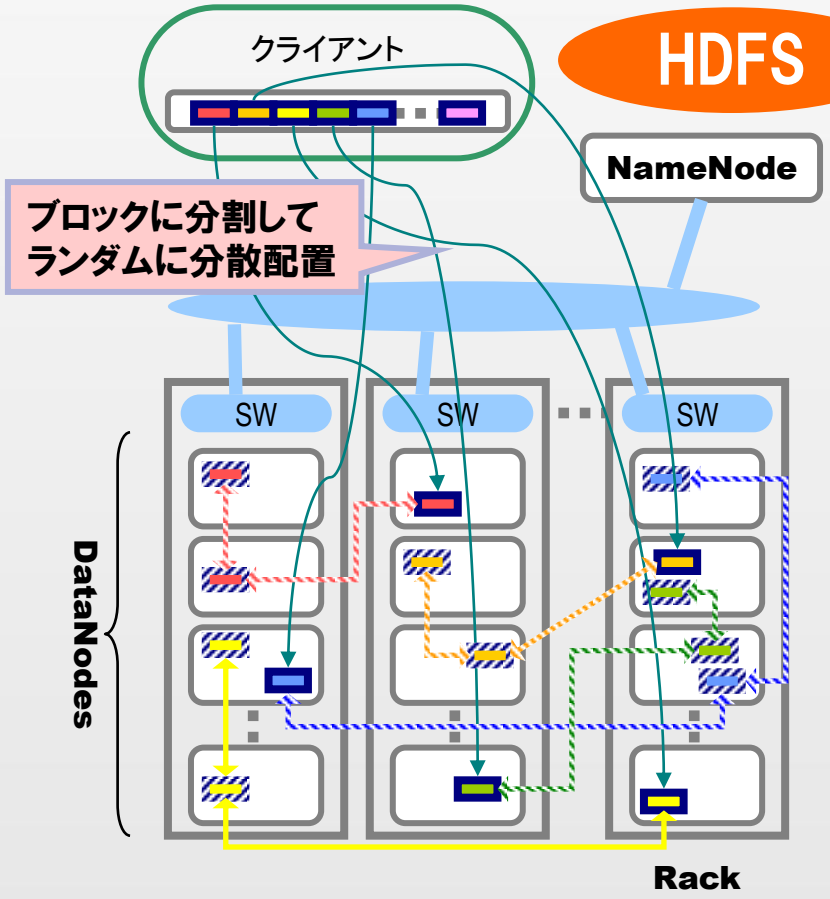
- Hadoopは、テラバイトやペタバイトクラスの大規模データのバッチ処理に向く
- リアルタイム処理、小さなデータの処理、更新処理には向かない
  - 分散処理のための前準備の時間として10秒～数十秒程度必要
  - RDBMSの適用が難しかった領域に適用可能だが、RDBMSの代替とはならない

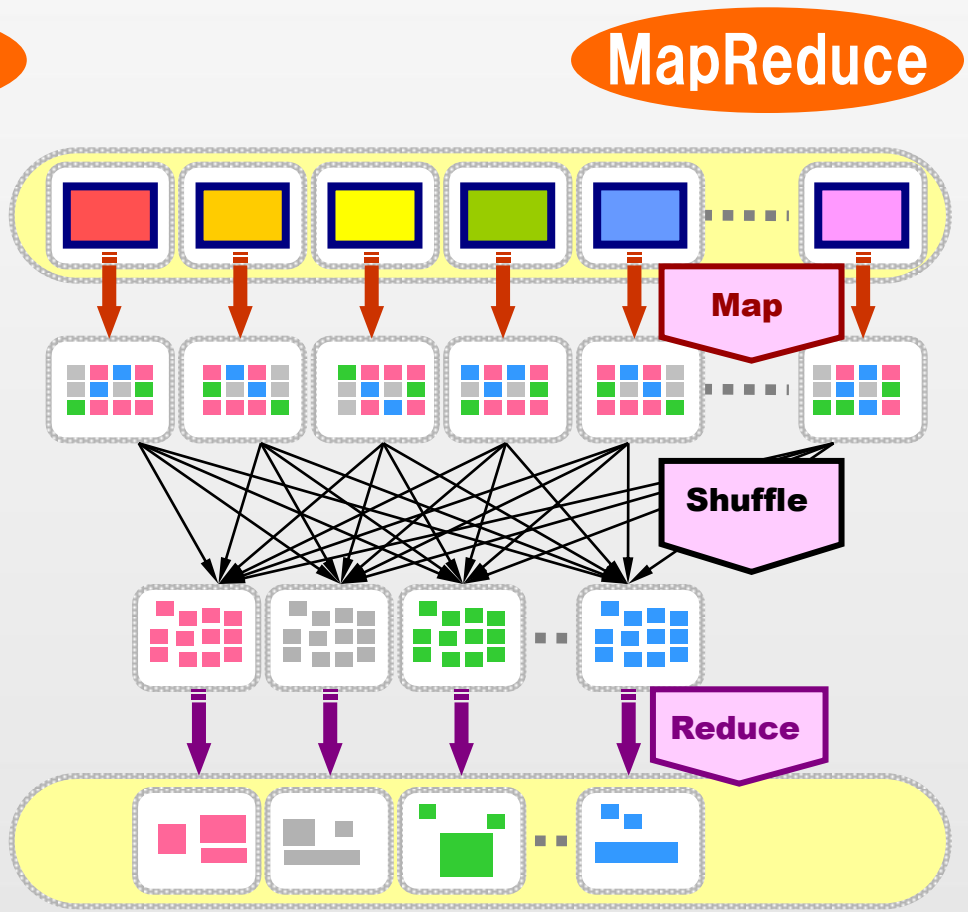
Hadoopの歴史

- Hadoopは1990年代後半から2000年初め頃にGoogleで開発された技術をベースとしている
  - 2003年に公開されたGoogle File System（GFS）
    - http://research.google.com/archive/gfs.html
  - 2004年に公開されたMapReduce
    - http://research.google.com/archive/mapreduce.html
- 2005年頃からDoug Cutting氏（当時Yahoo!）が中心となってHadoopの開発を開始
- 現在、Apache Software Foundationのプロジェクトとして開発が続けられている
  - http://hadoop.apache.org/

Hadoopのコンポーネント

- Hadoopは2つのコアコンポーネントから構成される
  - Hadoop Distributed File System（HDFS）：分散ファイルシステム
  - MapReduce：分散処理フレームワーク

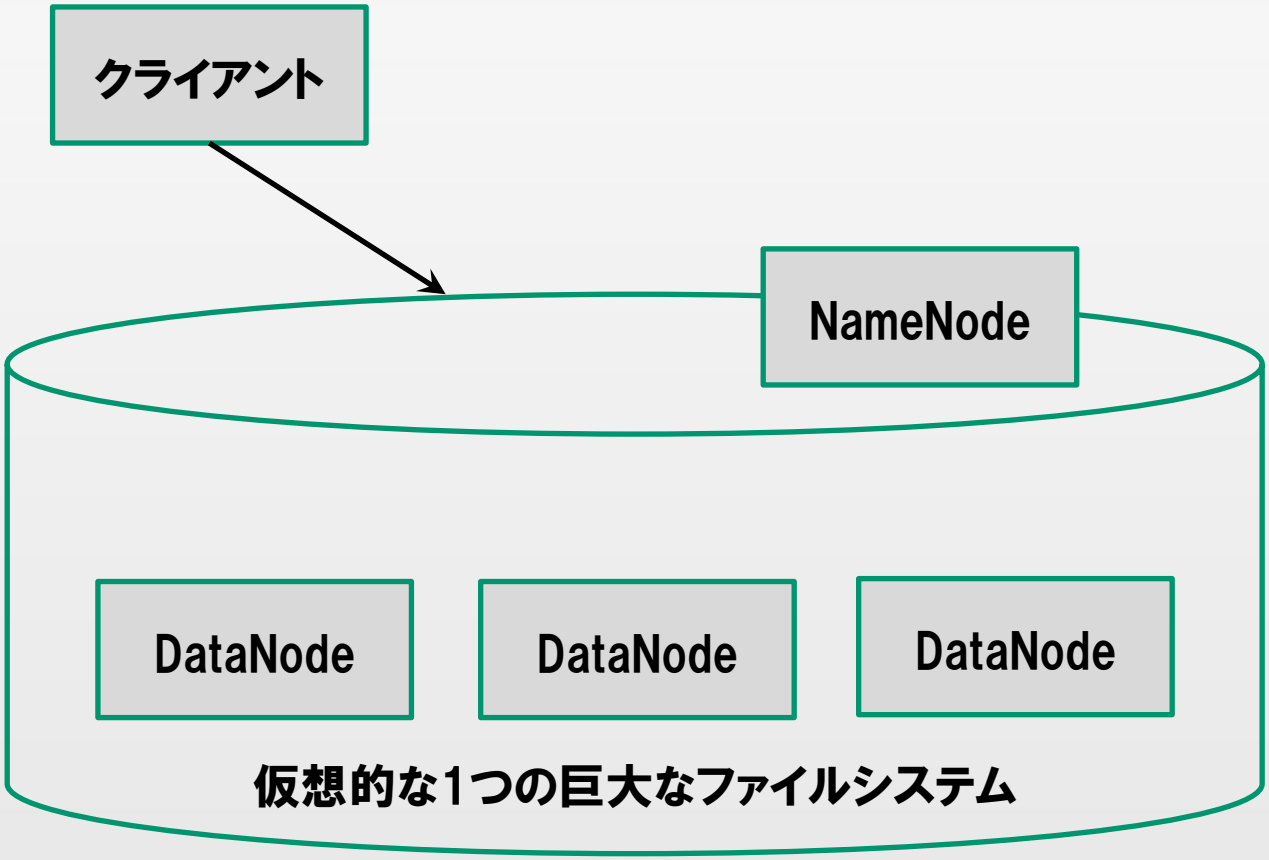




2. HDFSの概要

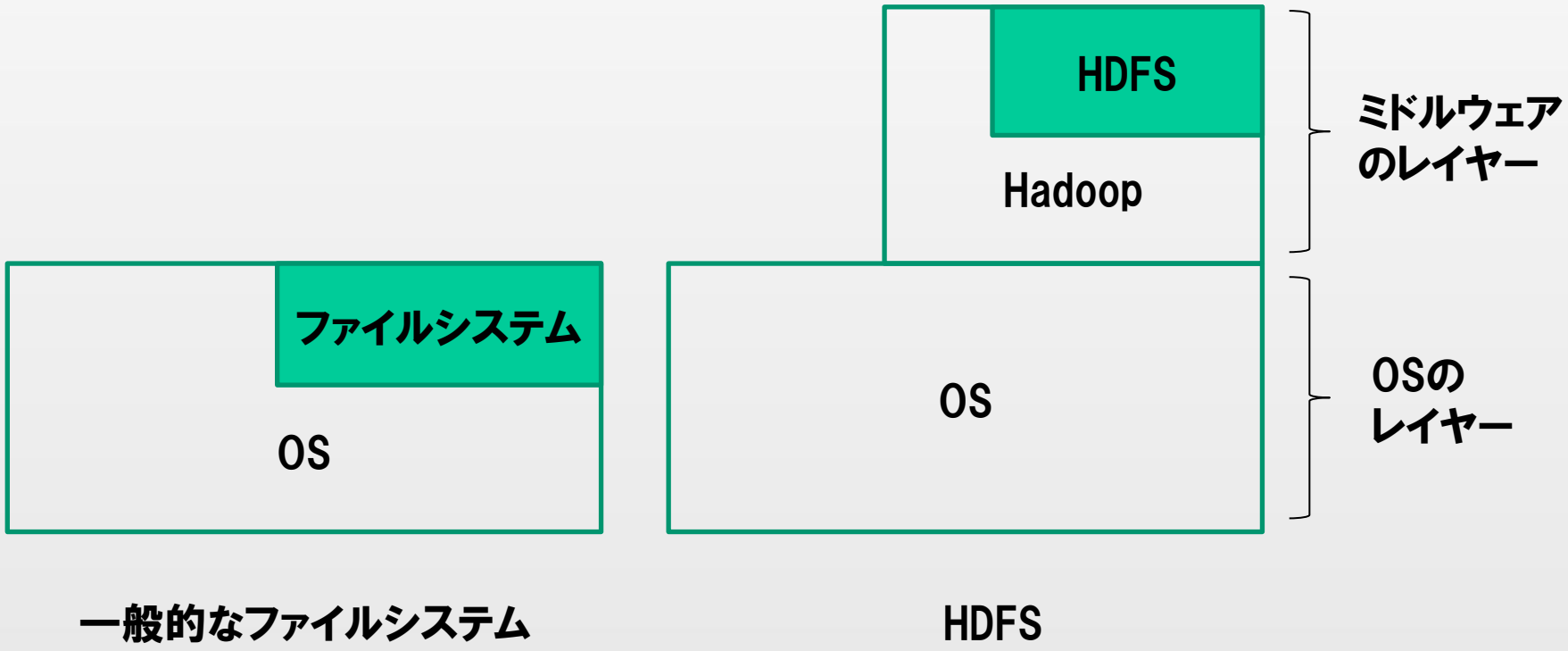
HDFSの概要

- 分散ファイルシステム
  - 複数のサーバで仮想的な1つの巨大なファイルシステムを構成する



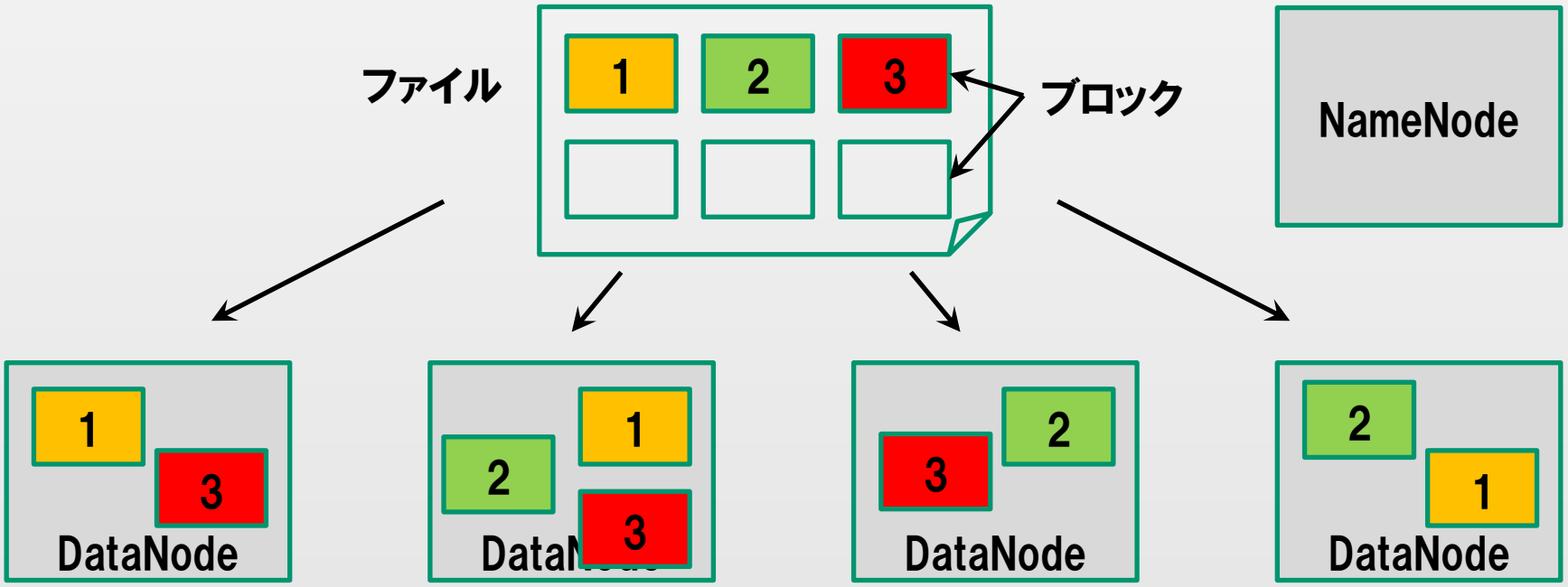
HDFSのレイヤー

- OSの1機能として提供されるファイルシステムではなく、OS上のミドルウェアのレイヤーで提供されるファイルシステム



ファイルとブロック

- ファイルは固定長のブロックに分割され、かつ各ブロックは複数個に複製され、それぞれ異なるDataNodeに格納される。複製数はデフォルトで3個で変更可
- OSのファイルシステムのブロックサイズは通常1～4KBだが、HDFSのブロックサイズはデフォルトで64MB(128MB等にも変更可)と非常に大きい。これはHDDのシーク時間の削減のため
- どのファイルのどのブロックがどのDataNodeに格納されているかという情報(メタデータ)をNameNodeが管理する



NameNodeの役割

- メタデータの管理
  - メタデータとは、ファイル、ブロック、DataNodeの関係を示す情報
  - パフォーマンスの観点からNameNodeのメモリ上で管理される
  - クラッシュに備えてディスク上に変更の記録を保持する
  - ファイル、ディレクトリ、ブロックに関するメタデータはそれぞれ150バイト程度
    - 1ファイルで300バイト程度(ファイル＋ブロック)、1億個のファイルで30Gバイト程度
- HDFSの使用状況の確認
  - HDFS全体や各DataNodeでの使用状況を管理
- クライアントからのHDFSの操作の受付
  - クライアントは最初にNameNodeにアクセスして対象ブロックがあるDataNodeのリストを取得し、その後、直接DataNodeにアクセスしてデータを取得する
- DataNodeの死活監視
  - DataNodeから一定間隔でハートビート(生存確認)パケットを受信する
  - ハートビートパケットを送信しなくなったDataNodeを故障とみなす
- 複製数(レプリカ数)の制御
  - ブロックのレプリカ数が設定された値よりも少なくなった場合や多くなった場合に、レプリカを作成したり削除したりする

HDFSの特徴

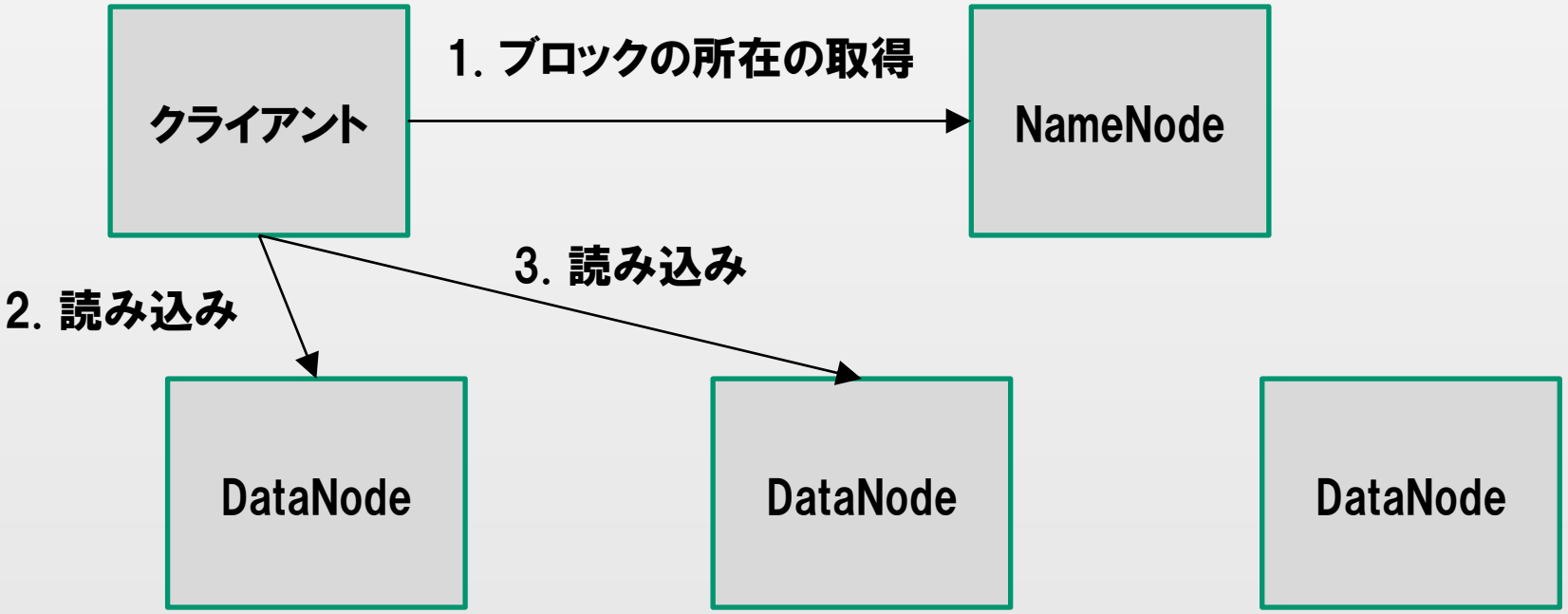
- 透過性
  - クライアントから見て、ファイルシステムの裏側で複数のサーバが動いていることを意識する必要はない
- 拡張性
  - サーバの台数を増やすことで、簡単にディスク容量を増やすことができる
- 信頼性
  - 1つのファイルを複数のブロックに分割し、それぞれのブロックを複数のサーバに冗長化して書き込むため、1つのサーバが壊れてもデータを失うことはない
- 制約事項
  - 高信頼性と高スループットを出す代わりに、以下の制約がある
    - シーケンシャルな読み込みを前提としており、ランダムな読み込みは想定していない
    - 一度書き込まれたデータは、その後読み込みだけを想定しており、データの追記はできるがランダムな更新を行うことができない
  - メタデータを管理するNameNodeが機能していないと、HDFSは利用できない
    - NameNodeが単一障害点となる
    - 可用性を高めるには、NameNodeを冗長化する工夫が必要

HDFSの特徴(続き)

- 用途
  - 巨大なファイルを格納するのに適している
  - 小さなファイルを大量に格納するのは、NameNodeのメモリを大量に消費するため、適していない
  - 一度格納したファイルは、更新はせず、読み出すだけのアクセスパターンに適している

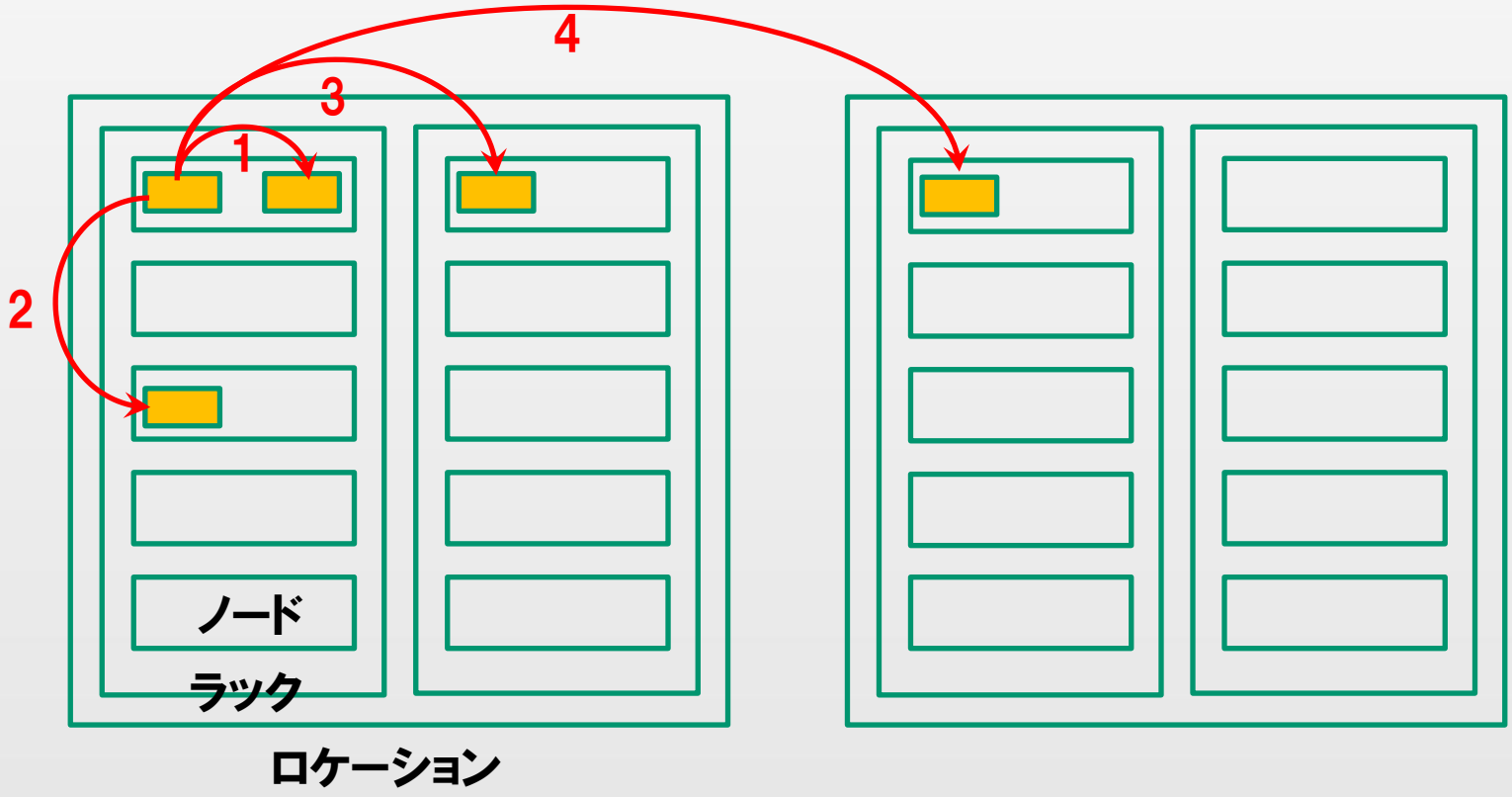
ファイルの読み込み

- 1. クライアントはファイルの一連のブロックがあるDataNodeのアドレスをNameNodeから取得する
- 2, 3. クライアントはブロックごとに最も近いDataNodeを選択して接続しファイルのブロックを読み込む(ノード間の距離は定義する必要あり)
- 必要に応じて上記を繰り返す



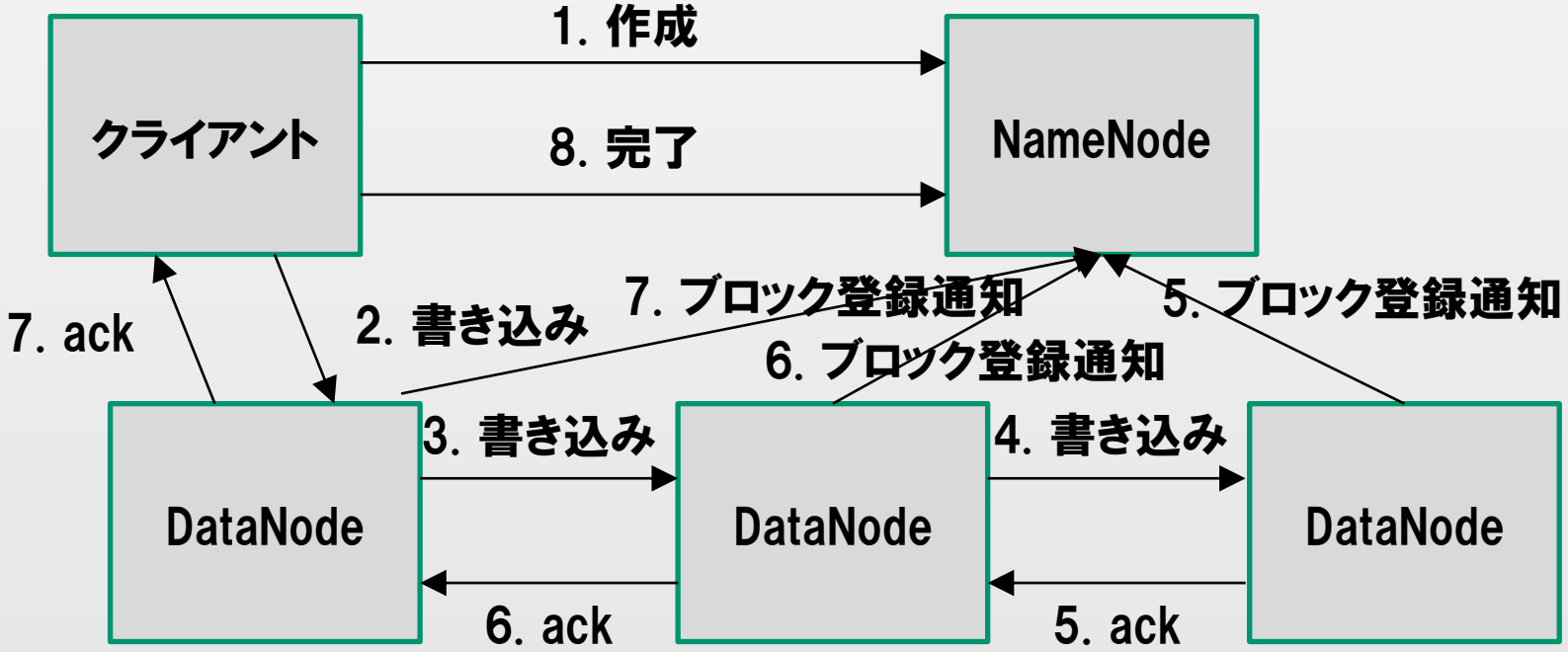
ノードの近さ

- ノードの近さの順
  - 1. 同一ノード内
  - 2. 同一ラック内の別のノード
  - 3. 同一ロケーション内の別のラックにあるノード
  - 4. 別のロケーションにあるノード



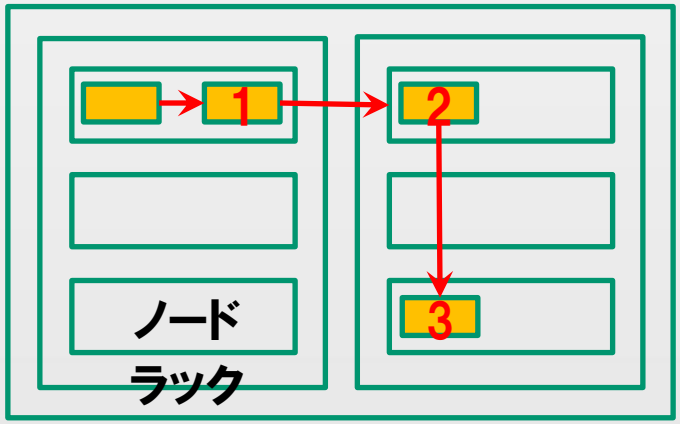
ファイルの書き込み

- 1. クライアントからの要求でNameNodeは新しいファイルのレコードを作成しブロックを割り当て、レプリカを配置するDataNodeのリストを返す
- 2. 3. 4. クライアントは分割データ(パケット)をDataNodeのリスト(パイプライン)のうち1番目のDataNodeに転送する。1番目のDataNodeはパケットを2番目のDataNodeに、2番目のDataNodeはパケットを3番目のDataNodeに転送する。
- 5. 6. 7. 8. ファイルの書き込みが完了したことを最終的にNameNodeに通知するとともに、各DataNodeはNameNodeにブロック登録通知を行う



レプリカの配置

- レプリカを配置するDataNodeの選択方法
  - 1つめのレプリカは、クライアントと同じノード上に配置される
  - クライアントがクラスタ外で動作している場合は、ランダムにノードが選択されるが、容量が使い切られそうなノードを選択しないようにする
  - 2つめのレプリカは、1つめのレプリカとは異なるラックをランダムに選んで配置する
  - 3つめのレプリカは、2つめのレプリカと同じラック内で、ランダムに選択された、2つめのノードとは別のノードに配置される
  - これ以降のレプリカは、クラスタ内のランダムなノードに配置されるが、同じラック内に多くのレプリカが配置されすぎないようにする



ファイルやディレクトリのパーミッション

- HDFS内のファイルやディレクトリにはPOSIXのように所有者／所有グループ、read／write／実行の権限を付与できる
- ただし、実行可能ファイルは存在しないため、実行権限は意味を持たず、setuid、setgidの概念（所有者や所有グループのふりをして実行できること）もない
- stickyビットの概念は存在し、stickyビットをONにしたファイルやディレクトリは、その所有者かスーパーユーザしか、それらを変更・削除することができない
- HDFSにおけるスーパーユーザは、OSのrootユーザではなく、NameNodeやDataNodeのデーモンを起動したユーザとなる
- 同じHDFSクラスタを構成するNameNodeとDataNodeのスーパーユーザは同じでなければならない
- 本講座の演習環境では、便宜的に、パーミッションのチェックを設定ファイルによりOFFにし（hdfs-site.xmlにおいて、dfs.permissions を false に設定）、任意のユーザがHDFS内の任意のファイルやディレクトを追加・削除可能にしている（通常はパーミッションをONにすること）

HDFSのアクセス方法

- コマンドラインからHDFSにアクセスするには、hadoop fsコマンドを使う
  - OSのコマンドと名前や意味が同じものが多い
  - hadoop fs の代わりに hadoop dfs でも可
- hadoopコマンドのヘルプメッセージの表示

```
$ hadoop
```

- hadoop fsコマンドのヘルプメッセージの表示

```
$ hadoop fs
```

HDFSのアクセス方法(続き)

- ローカルファイルシステムからHDFSへファイルをコピー

```
$ hadoop fs -copyFromLocal コピー元 コピー先
```

```
$ hadoop fs -put コピー元 コピー先
```

  - コピー元(ファイルまたはディレクトリ)は、ローカルファイルシステム上の絶対パス、またはカレントディレクトリからの相対パス
  - コピー先(ファイルまたはディレクトリ)は、HDFS上の絶対パス、またはHDFS上のユーザのホームディレクトリからの相対パス

- HDFSからローカルファイルシステムへファイルをコピー

```
$ hadoop fs -copyToLocal コピー元 コピー先
```

```
$ hadoop fs -get コピー元 コピー先
```

  - コピー元(ファイルまたはディレクトリ)は、HDFS上の絶対パス、またはHDFS上のユーザのホームディレクトリからの相対パス
  - コピー先(ファイルまたはディレクトリ)は、ローカルファイルシステム上の絶対パス、またはカレントディレクトリからの相対パス

HDFSのアクセス方法(続き)

- ローカルのファイルシステムからHDFSへファイルを移動

```
$ hadoop fs -moveFromLocal コピー元 コピー先
```

- HDFSからローカルのファイルシステムへファイルを移動

```
$ hadoop fs -moveToLocal コピー元 コピー先
```

- ディレクトリの内容一覧表示

```
$ hadoop fs -ls ディレクトリ名
```

- ディレクトリの内容一覧表示(再帰的)

```
$ hadoop fs -lsr ディレクトリ名
```

HDFSのアクセス方法(続き)

- **ファイルの内容表示**

```
$ hadoop fs -cat ファイル名
```

- **ファイルのコピー**

```
$ hadoop fs -cp コピー元 コピー先
```

- **ファイルの移動**

```
$ hadoop fs -mv 移動元 移動先
```

- **ファイルの削除**

```
$ hadoop fs -rm ファイル名
```

HDFSのアクセス方法(続き)

- ディレクトリの作成

```
$ hadoop fs -mkdir ディレクトリ名
```

- ディレクトリの削除(再帰的)

```
$ hadoop fs -rmr ディレクトリ名
```

- ディレクトリ配下のサイズの表示

```
$ hadoop fs -du ディレクトリ名
```

- ディレクトリ配下のサイズの表示(サマリ)

```
$ hadoop fs -dus ディレクトリ名
```

HDFSのアクセス方法(続き)

- ファイルのアクセス権の変更

```
$ hadoop fs -chmod [-R] モード ファイル名(またはディレクトリ名)
```

  - -R:指定したディレクトリ配下のファイルを再帰的に変更する
- ファイルのオーナーの変更

```
$ hadoop fs -chown [-R] オーナー名[:グループ名] ファイル名(またはディレクトリ名)
```

- ファイルのグループの変更

```
$ hadoop fs -chgrp [-R] グループ名 ファイル名(またはディレクトリ名)
```

- レプリカ数の変更

```
$ hadoop fs -setrep -w レプリカ数 [-R] ファイル名
```

  - -w:ファイルが指定したレプリカ数になるのを待つ
  - -R:指定したディレクトリ配下のファイルを再帰的に変更する


EDUCATION PROGRAM FOR TOP SOFTWARE ENGINEERS
TOP
SE
SOFTWARE
ENGINEERS

3. 演習：HDFSの操作

演習：HDFSを操作してみる

- 1. クライアントの仮想マシンにログインする。
- 2. 以下のコマンドを入力し、hadoopコマンドのヘルプメッセージを確認する。

```
$ hadoop
```

- 3. 以下のコマンドを入力し、hadoop fsコマンドのヘルプメッセージを確認する。

```
$ hadoop fs
```

- 4. HDFS上のルートディレクトリ配下のディレクトリ・ファイルの構成を確認する。

```
$ hadoop fs -lsr /
```

演習：HDFSを操作してみる（続き）

- 5. ローカルファイルシステム上にあるデータファイルを確認する。

```
$ cd /root/hadoop_exercise/01
$ ls -l enwiki
```

  - 以下の2つのデータファイルが存在
    - enwiki-long1.xml　：　150MB程度のファイル
    - enwiki-short1.xml　：　1MB程度のファイル
- 6. ローカルファイルシステムからHDFSへ、ディレクトリ配下のデータファイルをまとめてコピーする。

```
$ hadoop fs -put enwiki /enwiki
```

- 7. HDFS上にコピーされたデータファイルを確認する。

```
$ hadoop fs -lsr /enwiki
```

演習：HDFSを操作してみる（続き）

- 8. データファイルのサイズ、およびサイズのサマリを確認する。

```
$ hadoop fs -du /enwiki
$ hadoop fs -dus /enwiki
```

- 9. データファイルの内容を確認する。

```
$ hadoop fs -cat /enwiki/enwiki-long1.xml ¥
   | head -n 10
$ hadoop fs -cat /enwiki/enwiki-short1.xml ¥
   | less
```

  - データファイルのサイズが大きいため、出力をパイプでつなげてOSコマンドの more, less, head, tail等で部分的に確認する。

演習：HDFSを操作してみる（続き）

- 10. HDFS上でデータファイルをコピーする。コピー後、確認する。

```
$ hadoop fs -cp /enwiki /enwiki_backup
$ hadoop fs -lsr /enwiki_backup
```

- 11. HDFSからローカルファイルシステムへ、データファイルをコピーする。コピー後、確認する。

```
$ hadoop fs -get /enwiki_backup enwiki_backup
$ ls -l enwiki_backup
```

- 12. HDFS上およびローカルファイルシステム上それぞれで、コピーしたファイルを削除する。削除後、確認する。

```
$ hadoop fs -rmr /enwiki_backup
$ hadoop fs -ls /
$ rm -rf enwiki_backup
$ ls -lR
```

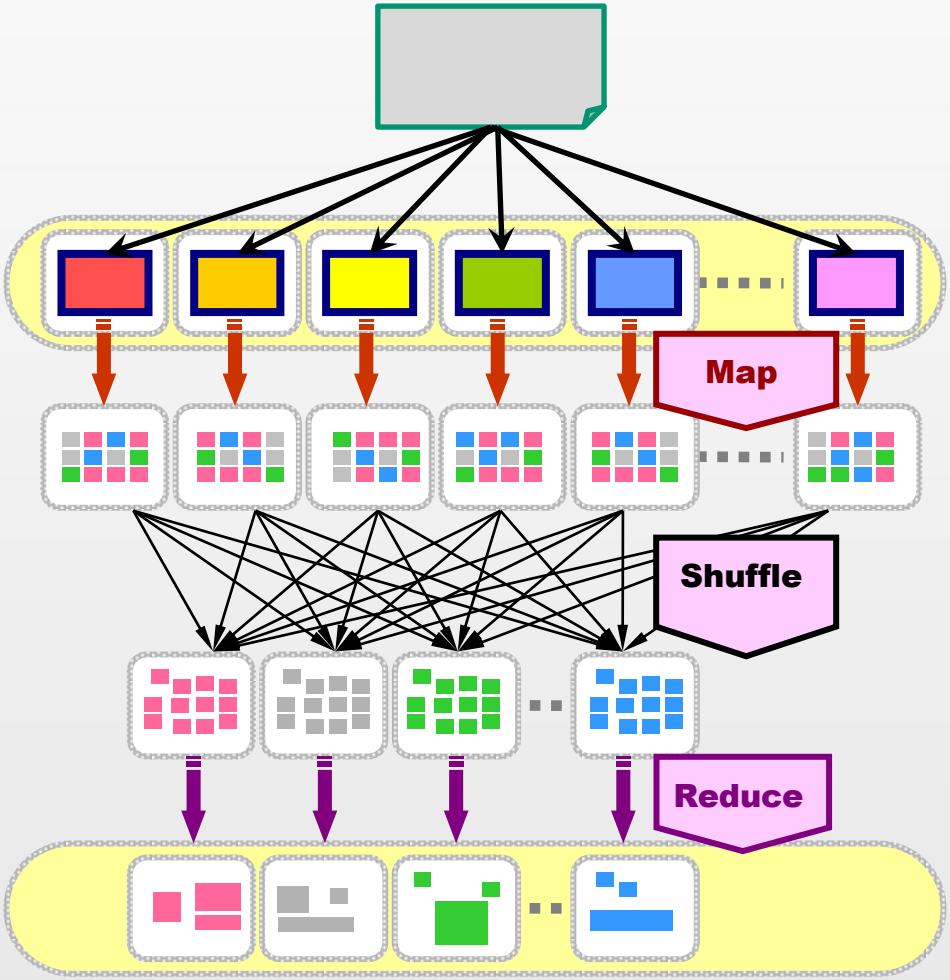
4. MapReduceフレームワークの概要

MapReduce

- 分散処理フレームワーク
  - Googleが検索インデックス作成のため考案
  - 数千台以上にスケールアウトしても性能向上することが知られている
- ユーザが定義したジョブを複数のタスクに分割し、クラスタ内の複数のノードに分散させて実行する仕組み
- 各ノードはできる限りそのノードに保存されたデータを処理する
  - データを処理のノードに移動させるのではなく、処理をデータのノードに移動させる
- 主に2つのフェーズから構成され、ユーザはこれらの処理を定義する
  - Map（抽出・変換）
  - Reduce（集約）
- MapとReduceの間にShuffleというフェーズが自動的に実行される
- MapReduceのプログラムは通常Javaで記述する
  - Hadoop Streamingを使って任意のスクリプト言語で記述することも可能
  - HiveやPigといった容易な方法で記述することも可能
  - 自動的にMapReduceに変換されて実行される

MapReduceの全体処理フロー

全体処理フロー図



入力データ

Map処理の入力
（分割されたデータ）

中間データ

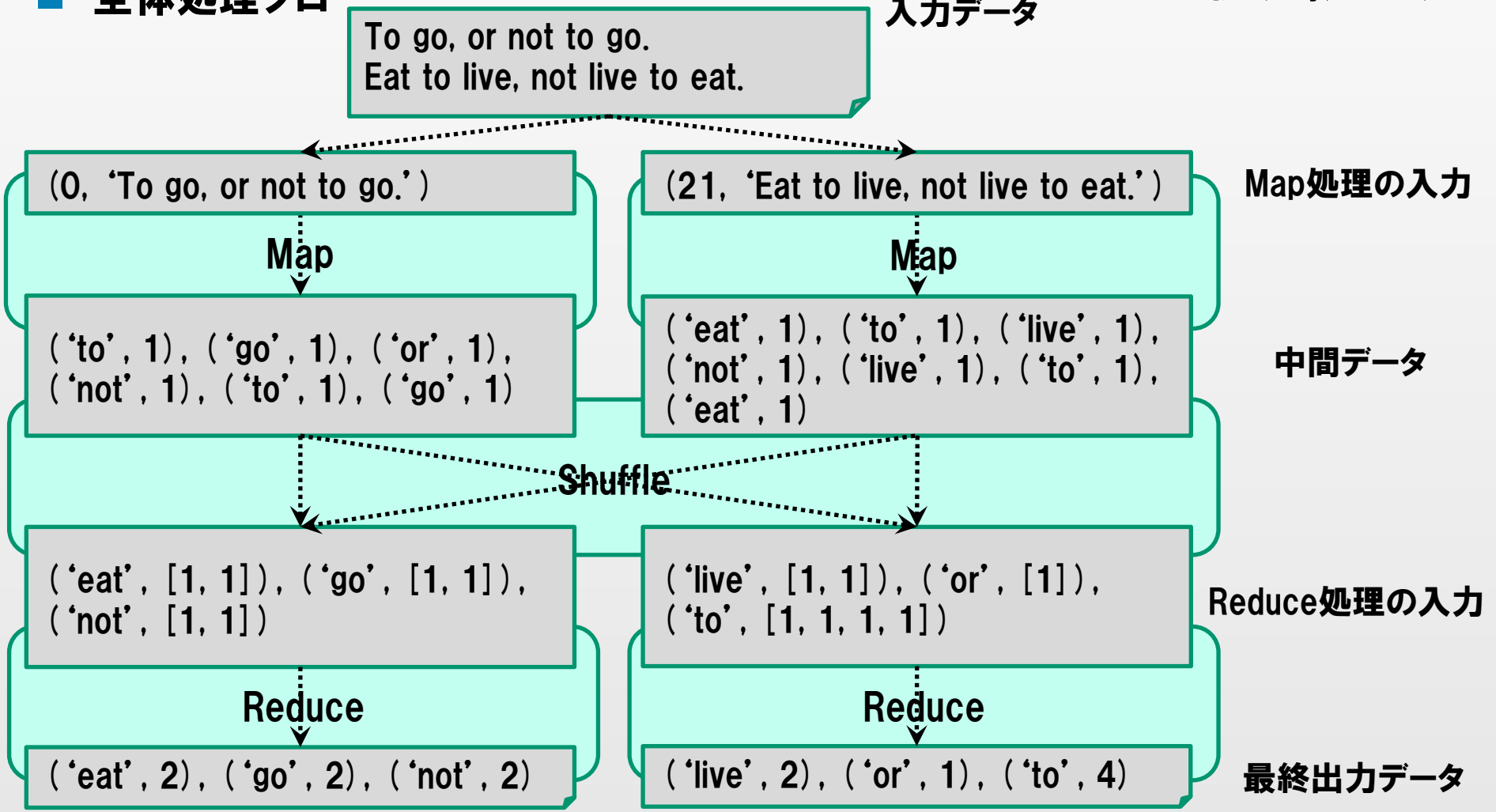
キーでソート

Reduce処理の入力

最終出力データ

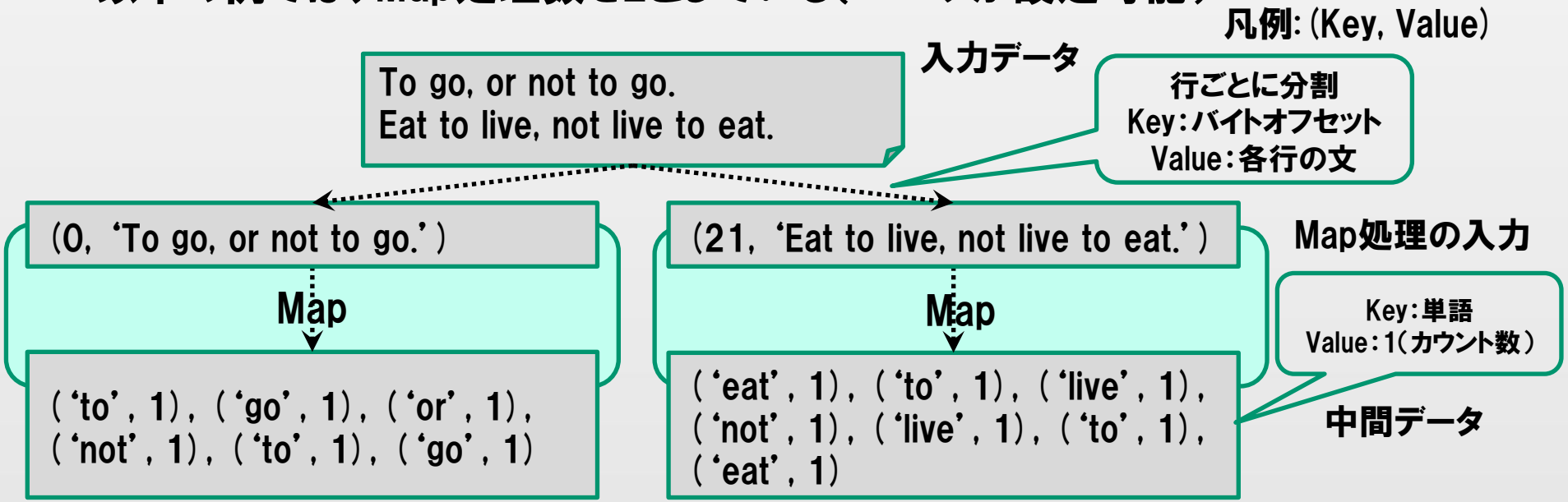
MapReduceの例 - WordCount

- WordCount：文章中の各単語の出現回数を数える
- 全体処理フロー



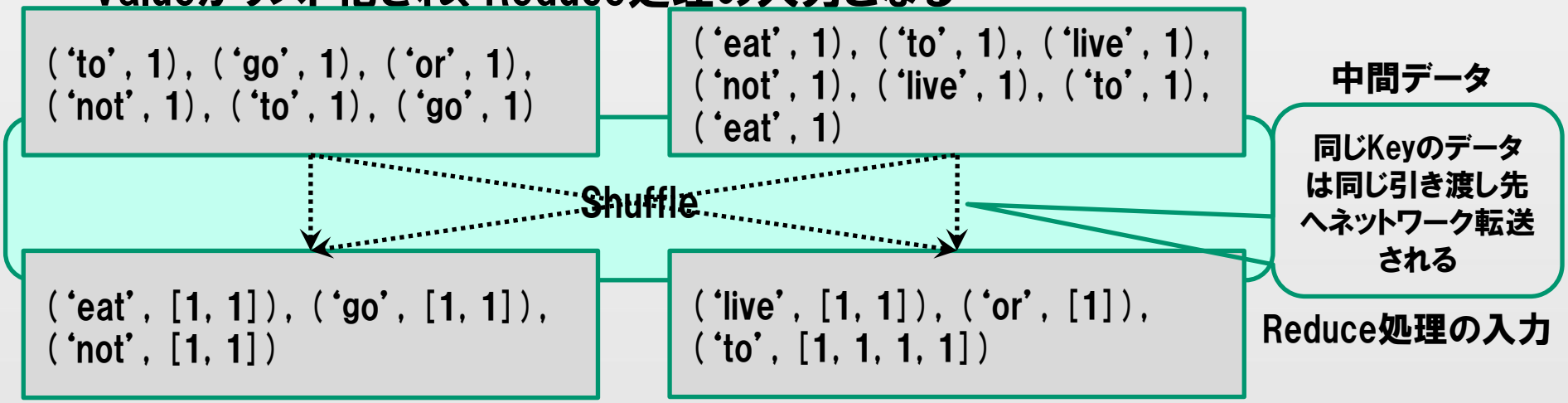
Mapフェーズ

- Mapフェーズでは、データを(Key, Value)のペア形式で読み込み、ユーザが定義したMap処理(抽出・変換処理)を実行し、(Key, Value)のペア形式で出力する
- 以下の例では、入力ファイルから1行ごとに読み取り、Key:ファイルの先頭からのバイトオフセット、Value:各行の文とし、各Map処理の入力とする
  - このKeyはMap処理では無視される
- 各Map処理では、文を単語に分け、1語ごとに、Key:単語、Value:1(カウント数)とした中間データを出力する
- 以下の例では、Map処理数を2としている(ユーザが設定可能)



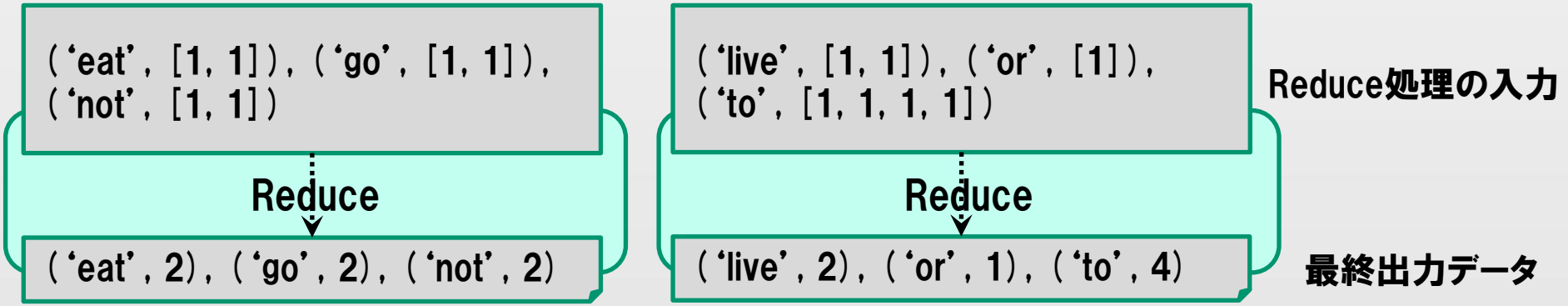
Shuffleフェーズ

- Shuffleフェーズは、フレームワークによって自動的に行われる
- 各Map処理が出力した中間データのうち、同じKeyのデータは、同じReduce処理の入力として引き渡される
- 引き渡し先は、デフォルトではKeyのハッシュ値をもとに決定される
  - ユーザが独自に定義することも可能(Partitioner:別途説明)
- 各データは、Map処理を行ったノードとReduce処理を行うノードが異なる場合、ネットワーク経由で転送される。このネットワーク転送がボトルネックとなりうる
  - 必要に応じてCombiner処理によってデータをあらかじめ集約しておく(別途説明)
- 引き渡し先のノードにおいて、各データはKeyでソートされ、同じKeyのデータはValueがリスト化され、Reduce処理の入力となる



Reduceフェーズ

- Reduceフェーズでは、データを(Key, Valueのリスト)のペア形式で読み込み、ユーザが定義したReduce処理(集約処理)を実行し、(Key, Value)のペア形式で出力する
- 以下の例では、各Keyにおいて、Valueのリストを加算している
- 同じKeyのデータは、前段のShuffleフェーズにより、同じReduce処理に割り当てられる
- 各Reduce処理において、入力となるデータは、前段のShuffleフェーズにより、Keyでソートされている
- 以下の例では、Reduce処理数を2としている(ユーザが設定可能)



入力スプリット

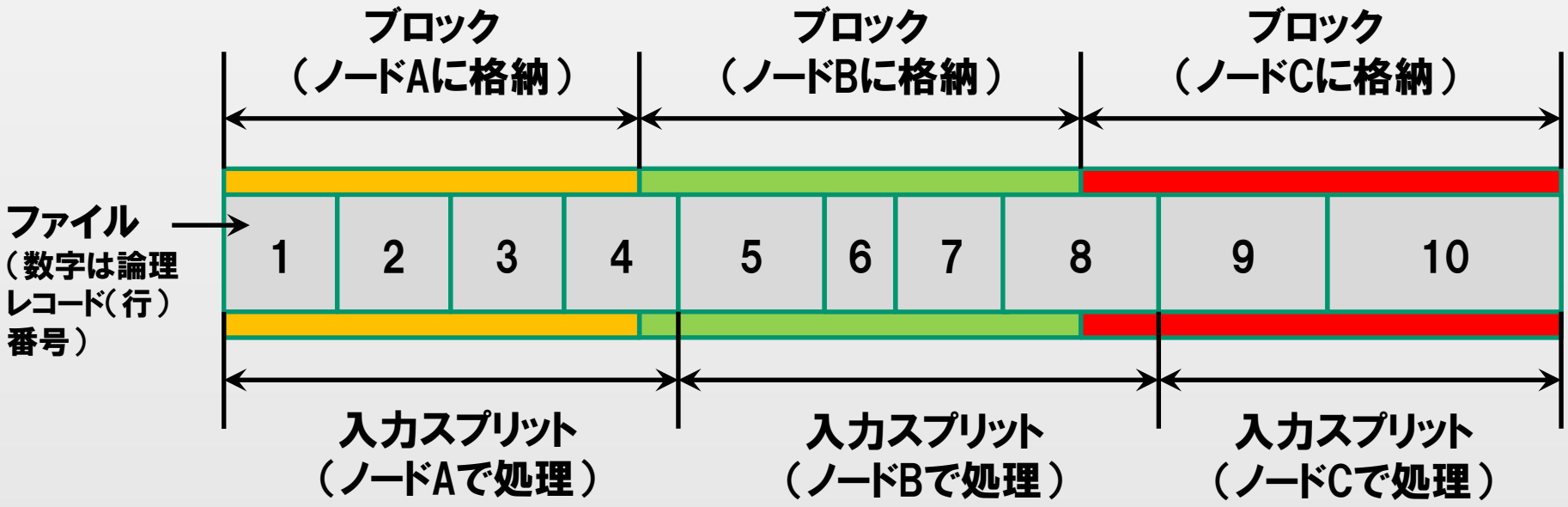
- 入力データは、入力スプリットと呼ばれる断片に分割される
- 各入力スプリットに対して、1つのMap処理が生成される
- Map処理では、入力スプリット中の各論理レコード(テキストファイルの場合は各行)に対して順次処理が実行される
- 可能な限り、入力データが格納されているノード上でMap処理が実行される

入力スプリットとHDFSブロック

- もし、入力スプリットがHDFS上で2つのブロックに渡るものになってしまうと、これらのブロックは別々のノードに格納されている可能性が高いため、入力スプリットの一部はネットワークを経由してMap処理を実行しているノードへ転送されなければならず、非効率
- 逆に、入力スプリットをブロックサイズに対して小さくしすぎれば、入力スプリットの管理やMap処理の生成に要するオーバーヘッドが大きくなり、非効率
- よって、入力スプリットのサイズと、HDFSのブロックサイズを、等しくするのが効率がよい

入力スプリットとHDFSブロックと論理レコード

- 論理レコード(テキストファイルの場合は行)は、HDFSブロックの境界をまたぐことがある
- 処理のためには、論理レコードを分割することはできない
- よって、入力スプリットは、HDFSブロックの境界よりも、論理レコードの境界の方を尊重する
- このため、入力データの一部分は、リモートのノードから転送される必要があり、多少のオーバーヘッドとなる

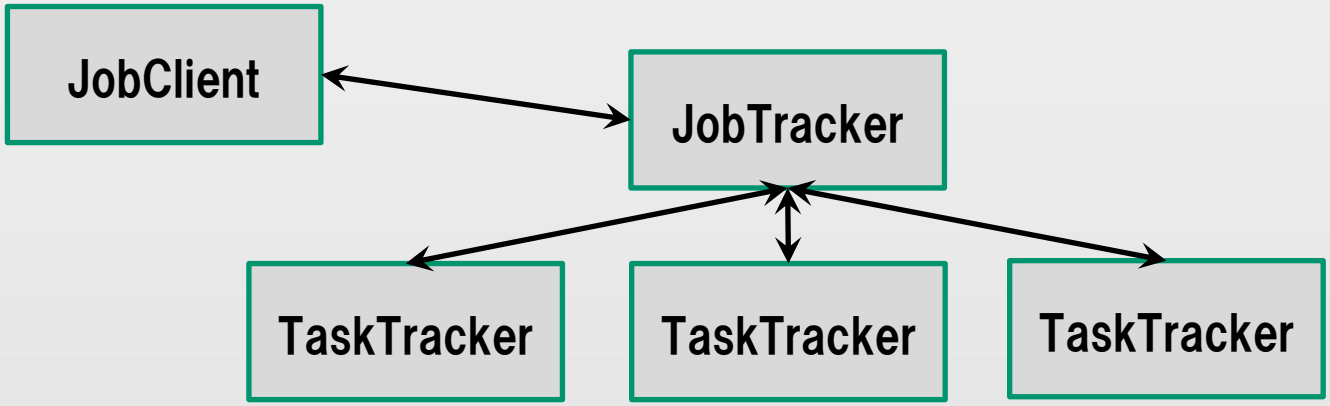


ジョブとタスク

- ジョブ
  - ユーザが実行を要求するMapReduce処理の単位
- タスク
  - ジョブを分割した、各Map処理や各Reduce処理の単位
  - それぞれ、Mapタスク、Reduceタスクと呼ぶ
  - MapReduceフレームワークによりジョブがタスクに分割され、各Mapタスクや各Reduceタスクは並列実行される

MapReduceフレームワークの構成要素

- JobTracker
  - ジョブの受付、TaskTrackerへのタスクの割り当て、ジョブやタスクの進捗と成否の管理、TaskTrackerの死活確認などを行う、クラスタの管理ノード
  - 1つのHadoopクラスタにつき、JobTrackerは1つ
    - JobTrackerは単一障害点となる
    - 可用性を高めるには、JobTrackerを冗長化する工夫が必要
- TaskTracker
  - タスクの処理やJobTrackerへの進捗報告、死活確認のためのハートビートの送信を行う、クラスタの計算ノード
- JobClient
  - Hadoopクラスタへジョブを投入するクライアント



スロット

- 各TaskTrackerは、スロットと呼ばれる処理単位を複数持ち、タスクを並列実行させることができる
  - Mapスロット数：Mapタスクの並列実行数
  - Reduceスロット数：Reduceタスクの並列実行数

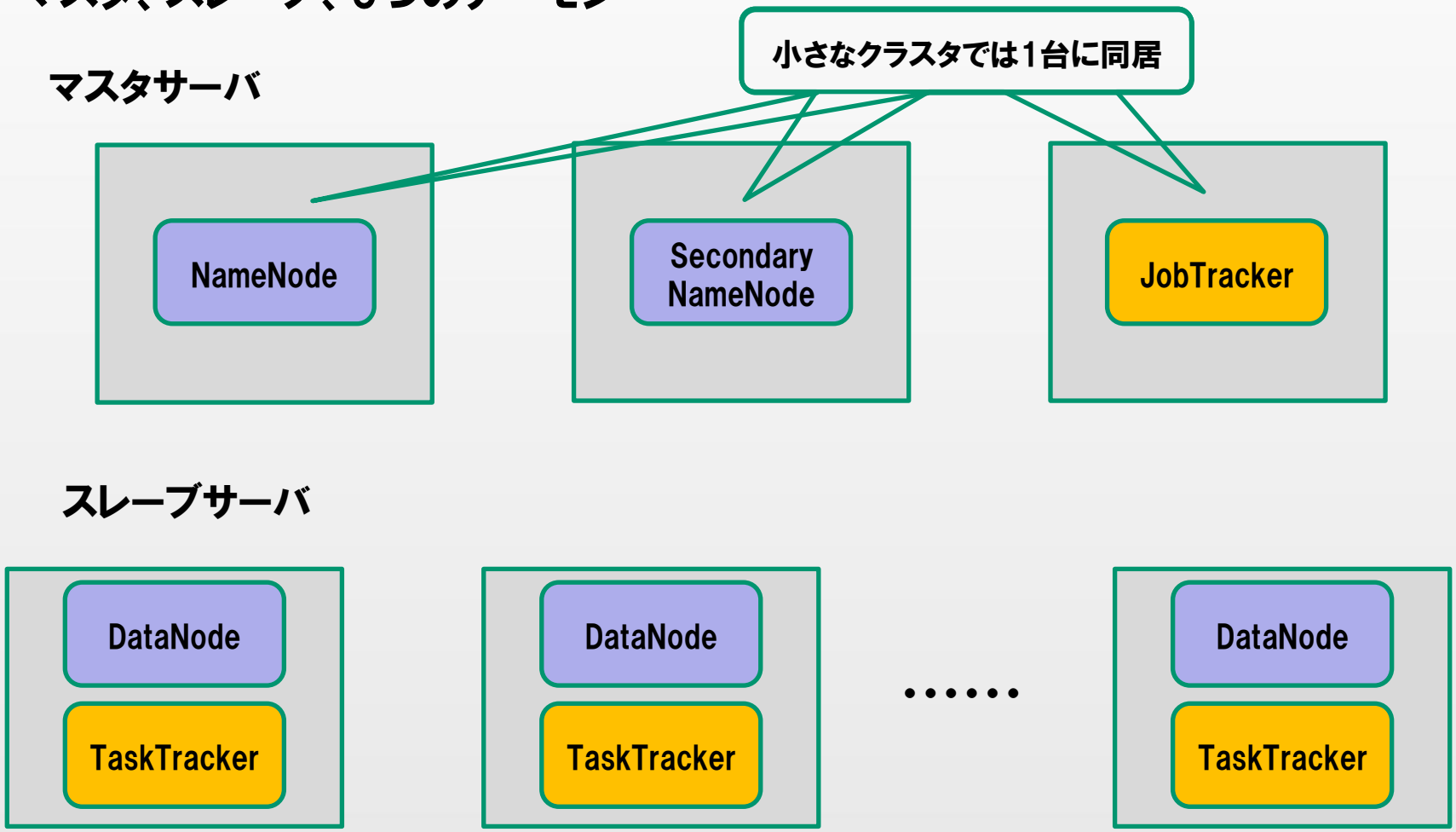
MapReduceのその他補足

- 各Map処理は、互いに通信することはなく、独立して実行される
- 各Reduce処理も、互いに通信することはなく、独立して実行される
- Map処理が出力する中間データは、余計なレプリケーションが行われないようHDFSではなく、Map処理が実行されたノードのローカルファイルシステムに一時保存され、ジョブの終了後に削除される
- Reduce処理が出力するデータは、信頼性を考慮してHDFSに保存される
- すべてのMap処理が完了してはじめて、Reduce処理が開始される。ただし、Map処理が出力した中間データは、すべてのMap処理が完了しないうちに、順次Reduce処理へ転送される
- 各Map処理や各Reduce処理のうち、明らかに遅い処理があると、同じデータを持つ他のノードでその処理が再実行される。これを投機的実行という
- Shuffleフェーズでネットワーク転送量を抑えるためにユーザがCombiner処理を定義したとしても、実際のCombiner処理の実行有無や回数は、フレームワークによって自動的に決められる

5. Hadoopクラスタの構成

Hadoopクラスタの構成

- マスタ、スレーブ、5つのデーモン



Hadoopクラスタの構成(続き)

- Hadoopは5つの独立したデーモンから構成される
  - NameNode
    - HDFSのメタデータを保持
  - SecondaryNameNode
    - HDFSのチェックポイント作業を行う
  - DataNode
    - 実際のHDFSデータブロックを格納
  - JobTracker
    - MapReduceジョブを複数のタスクに分割し、個々のタスクを各ノードに分配して、実行状況を管理
  - TaskTracker
    - それぞれのMapタスクやReduceタスクを実行し、結果をJobTrackerに通知
- 各デーモンは、それぞれ独立したJVM上で動作する

Hadoopの主な設定ファイル

- core-site.xml
  - Hadoopの共通的な設定を記述する設定ファイル
- hdfs-site.xml
  - HDFS関係の設定を記述する設定ファイル
- mapred-site.xml
  - MapReduce関係の設定を記述する設定ファイル

- 設定ファイルの格納先の例
  - /etc/hadoop/conf/ ディレクトリ内

Hadoopクラスタの典型的なハードウェアスペック

- Hadoopはコモディティのハードウェア上で動作するように設計されている
- 現在の典型的なハードウェアスペックの一例
  - CPU
    - クアッドコア Intel Xeon 2～3GHz×2（計8コア）
  - メモリ
    - マスタサーバ：48GB ECC RAM
    - スレーブサーバ：24GB ECC RAM
  - HDD
    - マスタサーバ：600GB SAS HDD×2（RAID1、実容量600GB）
    - スレーブサーバ：2TB SATA HDD×4（RAIDなし、実容量8TB）
      - 多重度が3の場合、クラスタ全体としてのデータの実容量は、8TB弱×スレーブサーバ台数／3
  - ネットワーク
    - 1Gbps（ギガビットイーサネット）

Hadoopクラスタの動作モード

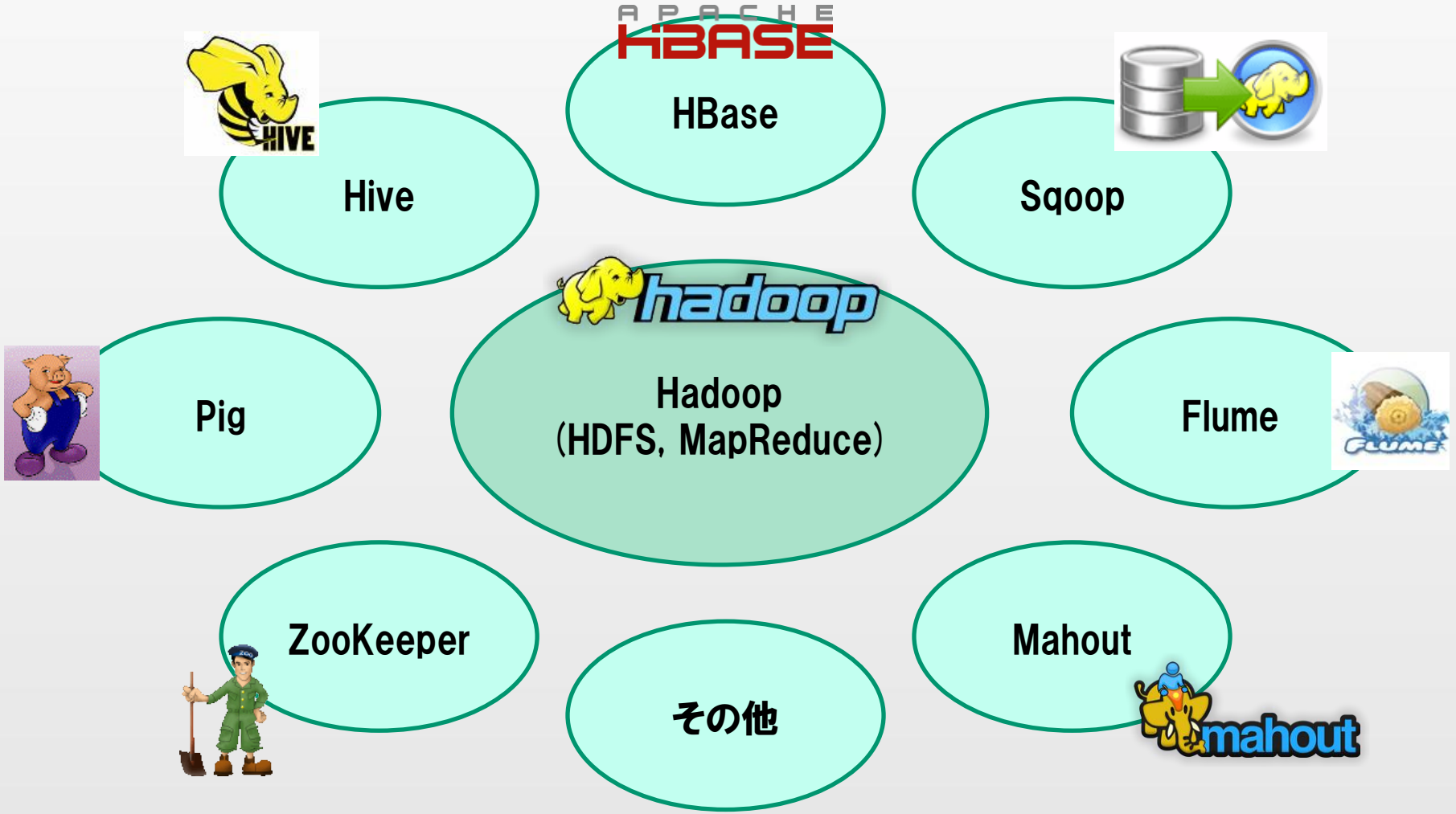
- 3つの動作モード

	ローカルモード	擬似分散モード	完全分散モード
利用目的	手元の環境でのMapReduce動作確認	手元の環境でのHDFS上でMapReduce動作確認	商用環境として利用する分散環境の構築
利用するサーバ台数	1台	1台	複数台
HDFS利用の有無	利用しない	利用する	利用する

6. Hadoopエコシステム

Hadoopエコシステム

- Hadoop(HDFS、MapReduce)を有効活用するために、Hadoopに関連する各種の周辺ソフトウェアを含めた、Hadoopエコシステムが形成されている



Hive

- 多くのアナリストにとって、MapReduceのJavaコードを書くのは難しいが、SQLクエリを書くことは容易
- Hiveは、そのようなアナリストのために、 SQLライクな言語であるHiveQLを用いて、データをMapReduceで分析することを可能とする
- HiveのクエリはMapReduceジョブに変換されて実行される
- RDBMSライクなテーブルを作成できる
- ただし通常のSQLやRDBMSに比べて制約は多い
- 当初Facebookで開発された
- 本講座の第10、11回の講義で詳しく扱う

Pig

- Hiveと同様にHadoop上のMapReduce処理をアドホックに(固定的でなくその場その場で手軽に)実行できるツール
- Pig Latinと呼ばれる処理言語で、データフローに沿った形で処理したい内容を定義する
- 定義した処理はPigがMapReduceジョブに変換した上で実行する
- データをどのように変化させたいかに着目して処理を定義できる
- 通常のMapReduceのコーディングに比べて少ない記述量で定義できる
- Yahoo!が主体となって開発された
- 本講座の第12回の講義で詳しく扱う

HBase

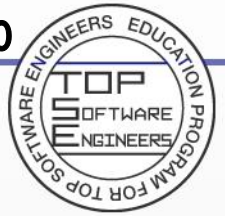
- HBaseはHDFS上にある列指向の分散型データベース管理システム
- Googleが開発したBigTableのクローン
- スケールアウト可能で、通常のRDBMSでは難しいテラバイトクラスのデータを扱うことができる
- ただし通常のRDBMSに比べて制約は多い
- 本講座の第13～15回の講義で詳しく扱う

その他

- ZooKeeper
  - Hbase等で使用されている、分散協調サービス
- Flume
  - 複数のシステムのログ等のデータを収集しHDFSに投入する
- Sqoop
  - RDBMSとHDFS間のデータのやりとりを実現する
- Mahout
  - 機械学習ライブラリ
- Oozie
  - 複数のMapReduceジョブのワークフローを定義

Hadoopディストリビューション

- Hadoopディストリビューションとは
  - Hadoopやその周辺ソフトウェア(Hadoopエコシステム)をパッケージにしたもの
- Hadoopディストリビューションを利用することの利点
  - Hadoopエコシステムを一括でインストールし利用することができる
  - 組み合わせて動作検証済みのため、ソフトウェア間の不整合に悩まされずに済む
  - サポートサービスを利用することができる(有償)
- Hadoopディストリビューションの一例
  - Cloudera's Distribution including Apache Hadoop (CDH)
    - http://www.cloudera.com/
    - 本講座の演習環境で利用
  - Hortonworks Data Platform
    - http://hortonworks.com/



7. Hadoopの利用事例

Hadoopの利用事例

- Yahoo!
  - 検索インデックス作成やレコメンデーションにHadoopを利用
  - 過去3年分のログデータ分析が従来の26日から約20分に短縮
  - 出典：Hadoop World 2009
  - http://www.slideshare.net/cloudera/hw09-hadoop-applications-at-yahoo
- VISA
  - 不正行為検出などの取引データ分析にHadoopを利用
  - 過去2年間で730億のトランザクション、36TBのデータが生成
  - 分析時間が従来の1ヶ月から13分に短縮
  - 出典：Hadoop World 2009
  - http://www.slideshare.net/cloudera/hw09-large-scale-transaction-analysis
- New York Times
  - HadoopとAmazon EC2を利用し、過去150年間の40.5万枚の巨大なTIFF画像、330万本のSGML記事、40.5万個のXMLファイルを、81万枚のPNG画像と40.5万枚のJavaScriptファイルに、36時間以下で変換
  - 出典：http://open.blogs.nytimes.com/2008/05/21/the-new-york-times-archives-amazon-web-services-timesmachine/

Hadoopの利用事例(続き)

- General Electric
  - Twitterやブログのデータから顧客の感性の分析にHadoopを利用
  - 従来の数十時間から数十分に短縮
  - 出典:Hadoop World 2010
  - http://www.cloudera.com/resource/hw10_ge_semtiment_analysis/
- CME Group
  - 取引のデータマイニングやシステムの遅延モニタリングにHadoopを利用
  - 出典: Hadoop World 2010
  - http://www.cloudera.com/resource/hw10_managing_derivatives_data_with_hadoop/
- 国立国会図書館
  - 検索システムのためのデータ変換、書誌同定処理、グループ化処理、インデックス作成にHadoopを利用
  - 出典:http://iss.ndl.go.jp/information/system/

まとめ

本講義で学んだ内容

- Hadoopの概要
  - 従来の大規模データ計算・分散処理の問題点、Hadoopのコンセプト、適用領域、歴史、コンポーネント
- HDFSの概要
  - ファイル、ブロック、NameNode、DataNode、ファイルの読み書きの仕組み、ノードの近さ、レプリカの配置、パーミッション、HDFSのアクセス方法
- 演習：HDFSの操作
  - hadoop fs コマンド
- MapReduceフレームワークの概要
  - Mapフェーズ、Shuffleフェーズ、Reduceフェーズ、入力スプリット、HDFSブロック、論理レコード、ジョブ、タスク、JobTracker、TaskTracker、JobClient、スロット
- Hadoopクラスタの構成
  - 5つのデーモン（NameNode、SecondaryNameNode、DataNode、JobTracker、TaskTracker）、主な3つの設定ファイル（core-site.xml、hdfs-site.xml、mapred-site.xml）、典型的なハードウェアスペック、3つの動作モード（ローカルモード、擬似分散モード、完全分散モード）

まとめ（続き）

- Hadoopエコシステム
  - Hive、Pig、HBase、その他、Hadoopディストリビューション
- Hadoopの利用事例
  - 検索インデックス作成、画像変換、ログデータ分析（レコメンデーション、不正行為検出、感性分析）、等