分散処理アプリ演習　第7回
MapReduceアプリケーションのテスト

（株）NTTデータ

講義内容

1. MapReduceアプリケーションのテスト
    1. MapReduceアプリケーションのテスト方法
    2. MRUnitを利用したMapReduceアプリケーションの単体テスト
    3. MRUnitを適用できない部分の単体テスト
    4. MapReduceアプリケーションのテストの進め方
    5. MapReduceアプリケーション開発のポリシー
    6. 演習：POSデータ分析アプリケーションのテスト
2. MapReduceアプリケーションの性能評価
    1. Hadoopの性能とは
    2. スケーラビリティに関する設定
    3. スケーラビリティを活用できない処理パターン
    4. 性能としてチェックする項目
    5. 処理時間の見積もり方
3. レポート課題

1. MapReduceアプリケーションのテスト方法

MapReduceアプリケーションでのテスト方法

- MapReduceアプリケーションをテストする場合、以下の方針となる
  1. mapメソッドやreduceメソッドでの実装が適切であるか
  2. Shuffle（Partitioner）で適切にデータが振り分けられるか
  3. MapReduceジョブ設定は適切であるか

- MapReduceアプリケーションのテストとしては、以下の通りに進める
  - 単体テスト：メソッドや設定が妥当であるかどうか
  - 結合テスト：MapReduceジョブとして処理されるかどうか

MRUnitを利用した単体テスト

- mapメソッドやreduceメソッドの実装をテストする仕組みとしてMRUnitが存在する

- MRUnitは、mapメソッドやreduceメソッドの実装が適切であることを確認するために利用する単体テスト用のツールである
  - MRUnitは、JunitをMapReduceアプリケーションに適した形にしたものである

- MRUnitによるMapReduceジョブのテストは、以下のクラスを利用する
  - MapDriver：Mapクラスのテスト用ドライバクラス
  - ReduceDriver：Reduceクラスのテスト用ドライバクラス
  - MapReduceDriver：MapReduceジョブ通しでのテスト用ドライバクラス

MRUnit：http://incubator.apache.org/mrunit/

MRUnitの利用方法

- MRUnitは、以下の手順で利用できる（Eclipseを利用する場合）
  1. 以下のWebサイトよりMRUnitのパッケージを入手
     - http://www.apache.org/dyn/closer.cgi/incubator/mrunit/
  2. 入手したMRUnitをJarファイルをEclipseでのライブラリに追加する
     - 「プロジェクト」→「プロパティ」→「Javaのビルド・パス」→「ライブラリー」タブ
     - 外部Jarの追加にてMRUnitのJarファイルを追加
  3. テスト用ドライバクラスを作成
     - Mapクラス用テストドライバ
     - Reduceクラス用テストドライバ
  4. MRUnitによるテストの実行
     - テスト用Javaファイルの指定（右クリック）→「実行」→「JUnit テスト」
  5. テスト結果の確認
     - テスト結果がすべてOKであるか確認する
     - NGの場合は、実装、テストの入出力が妥当か確認する

MRUnitでのテストドライバクラス

- MRUnitでのドライバは以下のような形で実装する

```
public class TestSampleMR extends TestCase {
  private MapDriver<LongWritable, Text, Text, Text> mDriver;
  private Mapper<LongWritable, Text, Text, Text> mapper;
  private ReduceDriver<Text, Text, Text, LongWritable> rDriver;
  private Reducer<Text, Text, Text, LongWritable> reducer;
  private MapReduceDriver<LongWritable, Text, Text, Text, Text, LongWritable> mrDriver;

  @Before
  public void setUp() {
    mapper = new SampleMRMapper();
    mDriver = new MapDriver<LongWritable, Text, Text, Text>(mapper);
    reducer = new SampleMRReducer();
    rDriver = new ReduceDriver<Text, Text, Text, LongWritable>(reducer);
    mrDriver =
      new MapReduceDriver<LongWritable, Text, Text, Text, Text, LongWritable>();
    mrDriver.setMapper(mapper);
    mrDriver.setReducer(reducer);
  }

  @Test
  public void testSampleMapper() {
    mDriver.withInput(new LongWritable(1), new Text("Abc"));
    mDriver.withOutput(new Text("A"), new Text("abc"));
    mDriver.withOutput(new Text("B"), new Text("ABC"));
    mDriver.runTest();
  }
```

テスト準備

Mapテスト

MRUnitでのテストドライバクラス（続き）

- MRUnitでのドライバは以下のような形で実装する

```
  @Test
  public void testSampleReducer() {
    List<Text> values = new ArrayList<Text>();
    values.add(new Text("ABC"));
    values.add(new Text("DEF"));
    rDriver.withInput(new Text("A"), values);
    rDriver.withOutput(new Text("A"), new LongWritable(2));
    rDriver.runTest();
  }
  @Test
  public void testMapReduce() {
    mrDriver.withInput(new LongWritable(1), new Text("WxyZ"));
    mrDriver.addOutput(new Text("A"), new LongWritable(1));
    mrDriver.addOutput(new Text("B"), new LongWritable(1));
    mrDriver.runTest();
  }
}
```

Reduceテスト

MapReduceとしてのテスト

MRUnitで処理できないテスト項目

- MapReduceジョブのテストのうち、MRUnitで対応できないものは以下の通りである
  - Partitionerの動作確認
  - WritableComparatorでのWritableの比較
  - RecordReader/RecordWriterのテスト
  - MapReduceジョブの設定状況の確認
  - DistributedCacheのテスト
  - HDFSなど他リソースとの連携確認
  - 複数の入力に対して出力を求めるテスト（1つの入力に対して複数の出力のテストは可能）

- これらのテストについては、JUnitなどの他の単体テストツールを利用することになる

MRUnitを適用できない部分の単体テスト

- 例：Partitionerのテストは以下のように記述する

```
private Partitioner<Text, IntWritable> partitioner;

@Before
public void setUp() {
  partitioner = new SamplePartitioner();
}

@Test
public void testPartitioner() {
  int p1 = partitioner.getPartition(new Text("A"), new IntWritable(1), 3);
  assertEquals(p1, 0);

  int p2 = partitioner.getpartition(new Text("B"), new IntWritable(2), 3);
  assertEquals(p2, 2);
}
```

MapReduceアプリケーションのテストの進め方

- MRUnitやJUnitでテストが完了した後は、MapReduce環境にてテストを続ける
  - LocalJobRunnerモード
    - mapred.job.tracker プロパティの値を local に変更
    - ローカル環境でHadoopデーモンを利用しないMapReduceジョブを実行
  - 擬似分散モード
    - 1台のサーバ上にHadoopに関するデーモンを立ち上げてテスト
    - fs.default.name プロパティに hdfs://localhost/ を設定して他ノードから起動させない
    - mapred.job.tracker プロパティに localhost を設定して他ノードからアクセスさせない
    - MapReduceフレームワークでのテストを適用することで、Partitionerの挙動など基礎的な挙動を確認できる
  - 完全分散モード
    - 小規模のデータを利用してテスト
    - ノード間を連携した処理が適切に実行されているか、処理に偏りが発生しないかを確認することができる

演習: POSデータ分析アプリケーションのテスト

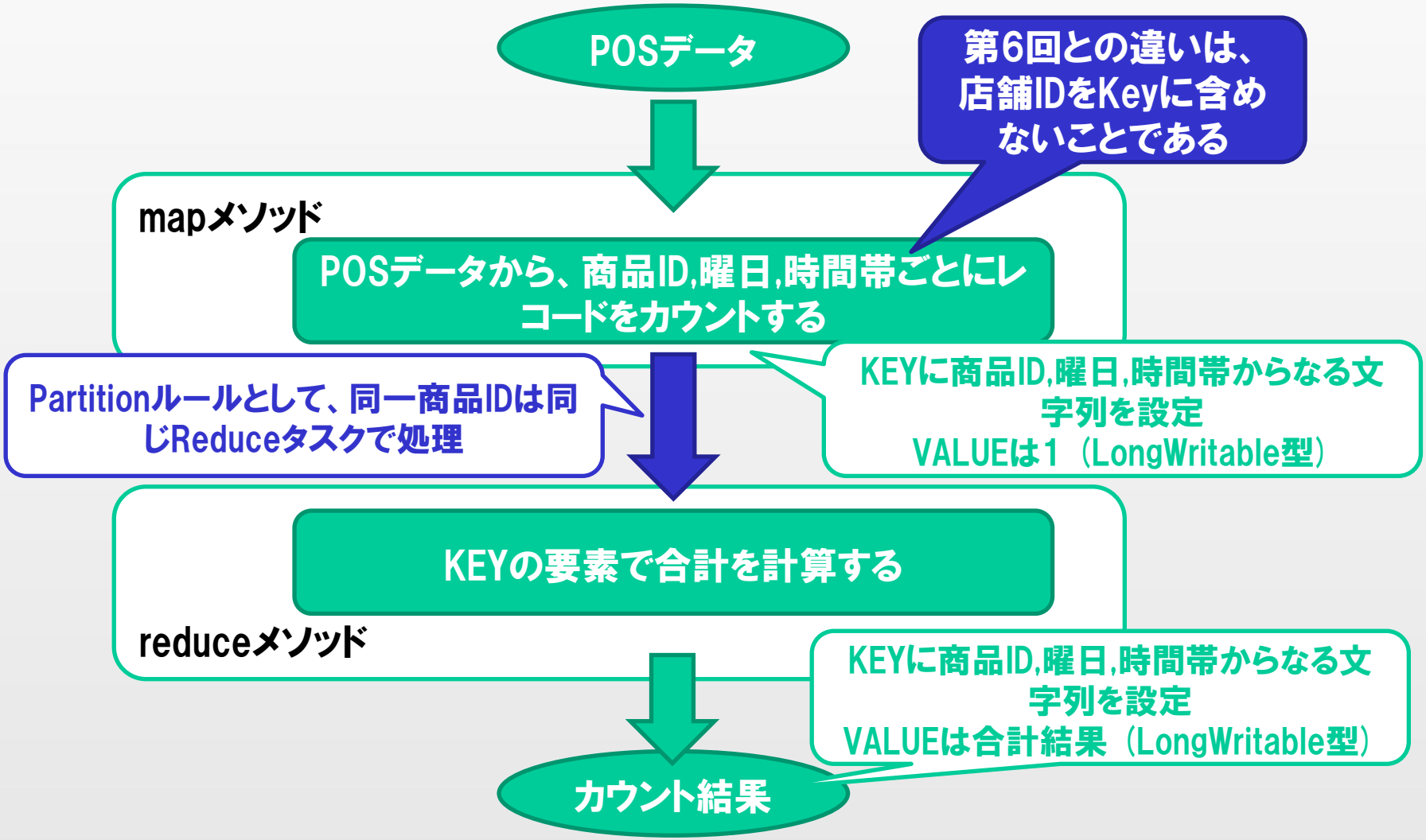
- POSデータ集計処理のソースコードを利用してMRUnitとJUnitでテスト項目を作成する

- テスト対象
  - MRUnit : POSCountGoodsJobMapper, POSCountGoodsJobReducer
  - JUnit : POSCountGoodsJobPartitioner

- MRUnitは、一部作成されているテストケースを確認しながら、いくつかテストパターンを作成する
- Partitionerのテストは、JUnitを利用してテストケースを作成する
  - 参考 : P.10

演習: POSデータ分析アプリケーションのテスト

- POSデータ集計処理の流れを以下に示す



演習環境 - データ類

- 演習で利用する資材は、以下の通り配置されている
  - /root/hadoop_exercise/06/ 以下にデータは格納されているものを利用

- POSデータ：posdata.tar.gz（pos-dataディレクトリに格納）
  - 第6回で未実施の場合POSデータは、以下のコマンドを実行してHDFS上に配置すること

```
## POSデータを展開する
$ cd /root/hadoop_exercise/06/pos-data/
$ tar xvzf posdata.tar.gz
## POSデータをHDFS上に格納する
$ hadoop fs -mkdir hadoop_exercise/06/pos-data
$ hadoop fs -put posdata.csv hadoop_exercise/06/pos-data/
## POSデータをHDFS上に配置されているか確認する
$ hadoop fs -ls hadoop_exercise/06/pos-data/
```

演習環境 - ソースコード

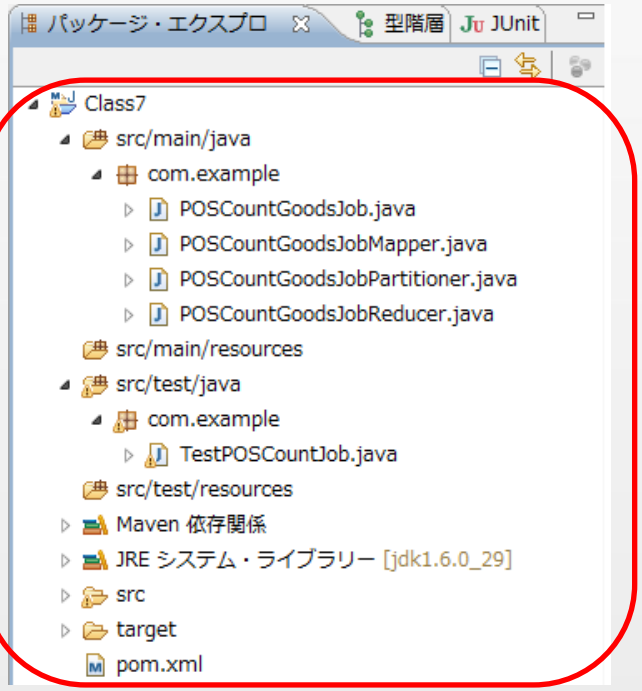
- 演習で利用する資材は、以下の通り配置されている
  - /root/workspace/Class7/ 以下にデータは格納されている

- POSデータ集計処理用クラス
  - POSCountGoodsJob：POSデータ集計処理MapReduceジョブクラス
  - POSCountGoodsJobMapper：POSデータ集計処理Map処理クラス
  - POSCountGoodsJobPartitioner：POSデータ集計処理Partitionerクラス
  - POSCountGoodsJobReducer：POSデータ集計処理Reducerクラス

- POSデータ集計処理用テストクラス
  - TestPOSCountGoodsJob：POSデータ集計処理テストクラス

演習環境 – Eclipseプロジェクト

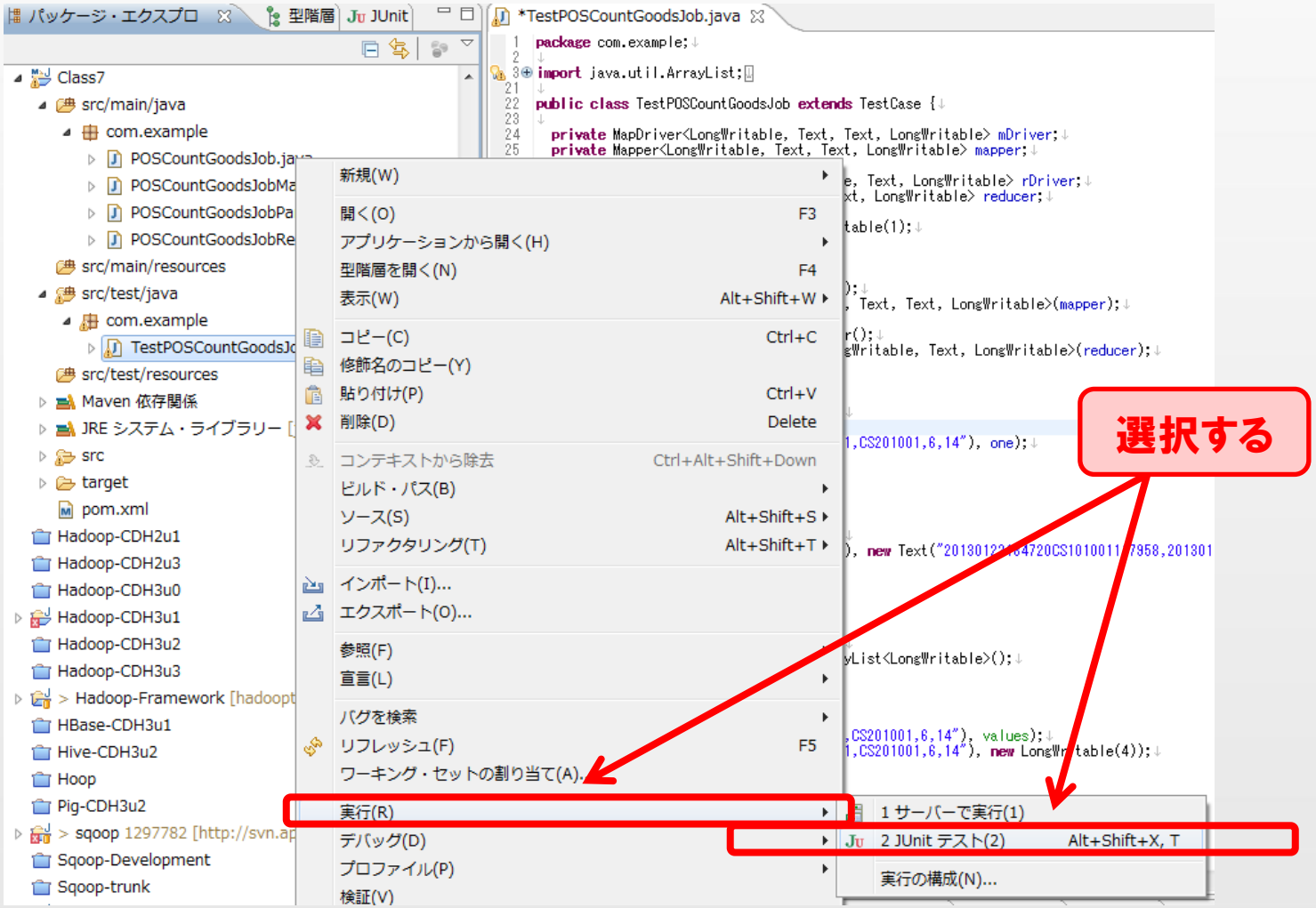
- EclipseのClass7プロジェクト内に、必要な資材がそろっている



「Class7」プロジェクト内の
「src/main/java」ディレクトリ内に
ソースコードが格納される
「src/test/java」ディレクトリ内に
演習対象の資材が用意されている

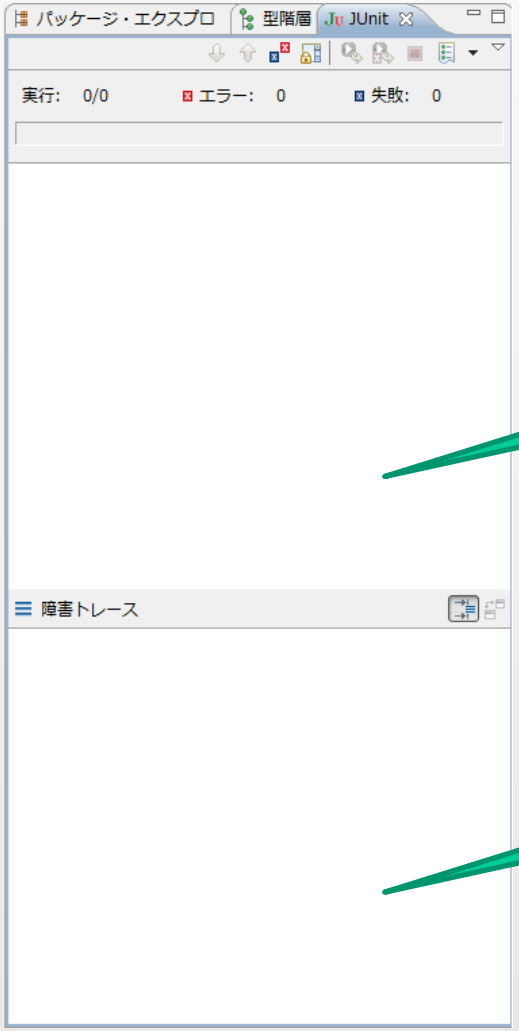
テスト実行方法

- テスト用ソースコードより [実行] → [JUnitテスト] を選択する



テスト結果の確認

- Eclipse上でテスト結果も確認できる



作成したテストケースでのテスト実行状況が表示される

失敗したテストケースでの失敗状況が記録される

アプリケーションの実行

- 第6回の演習（POSデータ分析処理）と同様の手順でMapReduceを実行する

2. MapReduceアプリケーションの性能評価

Hadoopの性能とは

- Hadoopは、MapReduceジョブを実行する際にそのスケーラビリティや並列処理のスループットが性能のポイントになる

- Hadoopのスケーラビリティとは、以下の2点がポイントとなる
  - **処理スロット数に関するスケーラビリティ**
    - 処理スロット数の変化と処理時間が比例の関係であること
  - **処理データ量に関するスケーラビリティ**
    - 処理データ量が変化と処理時間が比例の関係であること

スケーラビリティに関する設定

- Hadoopのスケーラビリティに関する設定は、以下の通りである

1. 処理スロット数に関するスケーラビリティ
   - TaskTrackerで起動するMapスロット数/Reduceスロット数を設定する
   - mapred.tasktracker.map.tasks.maximum：Mapスロット数の設定
   - mapred.tasktracker.reduce.tasks.maximum ：Reduceスロット数の設定
   - どちらもCPUコア数と同程度（または、1コア程度少ない）値に設定する

2. 処理データ量に関するスケーラビリティ
   - HadoopのMapタスクは、HDFS上のブロックに従い処理されるため、ブロックサイズに極端に満たないファイルをMapReduceで扱うと効率が悪い
   - 極端に小さいファイルは結合してブロックサイズ程度（よりも大きな）ファイルに変更する

スケーラビリティを活用できない処理パターン

- MapReduceのスケーラビリティを利用できないパターンは以下の通りである
  1. 全件データをソートする場合
     - Reduceで全件データを集約させるため、特定のReducerにデータがすべて集中
     - 1つのReducerしか実行しないため、スケーラビリティの恩恵を受けない
     - 処理データ量次第では、最初に最大値と最小値をもとめて、Partitionerを値幅ごとに設定して並び替える手法は考えられる
  2. Partitionerで設定したKey設定に関して、特定のReducerにデータが集中する場合
     - Partitionerでは、ある程度までの分散しか実現できない
       - 例：都道府県単位でのPartition設定
     - Partitionerによる分割の影響を受けないようなPartitionerの実装が必要
  3. MapReduceジョブから他のリソースにアクセスするような場合
     - 他のリソース（例: RDBMS）がスケーラブルな仕組みになっていない場合、一定数までの同時接続しか実現できず、スケーラビリティの恩恵を受けられない可能性がある
     - 他リソースの接続数や性能について考慮して実装すること

性能としてチェックする項目

- MapReduceジョブを実行する場合、性能面で以下のことを確認する
  - タスクの平均時間よりも突出して長いタスクが存在するか
    - JobTrackerのWeb画面より、タスクの平均時間、遅いタスクの処理時間を確認できる
    - 遅いタスクが存在する場合、カウンター情報などを確認し、処理時間が延びる実装が含まれていないか確認する
  - 特定のノードのリソース使用状況に偏りが無いか
    - 特定のノードでのみ処理に時間が掛かるような場合、リソース枯渇・競合が懸念される
    - ノードのプロセス状況などを確認し、異常が無いか確認する
  - Shuffleに時間が掛かる
    - Mapタスク数が多い場合、ShuffleでMap処理結果を取得する場合時間が掛かる
    - Mapタスクが多くならないようにHDFS上のデータを扱う
      - ブロックサイズの拡大
      - CombineFIleInputFormatの適用

処理時間の見積もり方

- MapReduceジョブを実行する場合、いきなり大規模データを利用して実行することは処理時間の観点で難しい
  - 何らかの小規模データを扱い、大規模データを実行する場合の処理時間を見積もる必要がある

- 見積もりのポイントとなる要素は以下の通りである
  - 1タスクでのMap処理時間（最長時間を考慮）
  - 1タスクでのReduce処理時間（最長時間を考慮）
  - Shuffleに要した時間
    - Reduceの開始タイミングをMap処理がすべて終了した後とすることで、正確なShuffleの時間も把握できる
  - Mapスロット数とReduceスロット数
  - 大規模データでのMap処理数とReduce処理数
    - どちらも概算でよい

処理時間の見積もり方

- 大規模データを実行する場合の処理時間は、以下の通りとなる

処理時間 = Map時間 + Shuffle時間 + Reduce時間
Map時間 = 切り上げ(Map処理数 / Mapスロット数) ×1タスクのMap処理時間
Shuffle時間 = Map処理数 / 小規模Map処理数 × 小規模Shuffle時間
Reduce時間 = 切り上げ(Reduce処理数 / Reduceスロット数)
× 1タスクのReduce処理時間
(赤字：小規模データでの試行による指標値)

- 1タスクのMap処理時間やReduce処理時間は、小規模実行時の最長時間を設定することで最悪値ベースで処理時間を見積もることができる
- 通常HadoopのMapReduceジョブは、Mapの進捗が 5%になるとShuffleが開始されるため、Shuffle時間は見積もり時間よりは短くなることが期待できる
  - mapred.reduce.slowstart.completed.maps プロパティ、デフォルト 0.05

まとめ

本講義で学んだ内容

- MapReduceアプリケーションのテスト
  - MapReduceアプリケーションのテスト方法
  - MRUnitを利用したMapReduceアプリケーションの単体テスト
  - MRUnitを適用できない部分の単体テスト
  - MapReduceアプリケーションのテストの進め方
  - MapReduceアプリケーション開発のポリシー
  - 演習：POSデータ分析アプリケーションのテスト
- MapReduceアプリケーションの性能評価
  - Hadoopの性能とは
  - スケーラビリティに関する設定
  - スケーラビリティを活用できない処理パターン
  - 性能としてチェックする項目
  - 処理時間の見積もり方

3. レポート課題

レポート課題：POSデータを用いたMapReduceアプリケーションの作成

- POSデータを用い、以下の条件に関するMapReduceアプリケーションを作成する

- 条件（以下の2つ以上を利用すること）
  - セッション単位でデータを操作する
  - POSデータに含まれる属性要素（年齢・性別など）を利用すること
  - 商品情報にて商品コード（商品IDの先頭2桁）を利用すること

- 提出物
  - ソースコード（コメントを含めること、どの条件を利用したか示すこと）
  - 実行用JARファイル
  - 実行結果
  - 未完成であっても部分点を出すので提出すること
- 提出方法：上記のファイル群をtar.gzまたはzipで1つにまとめてLMSに提出
- 提出期限：当日周知
- 質問先：当日周知