クラウド実践（第5-6回）
MapReduceプログラミング

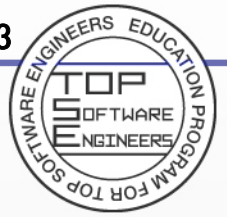
第2版

2012年4月28日

トップエスイープロジェクト
田辺良則

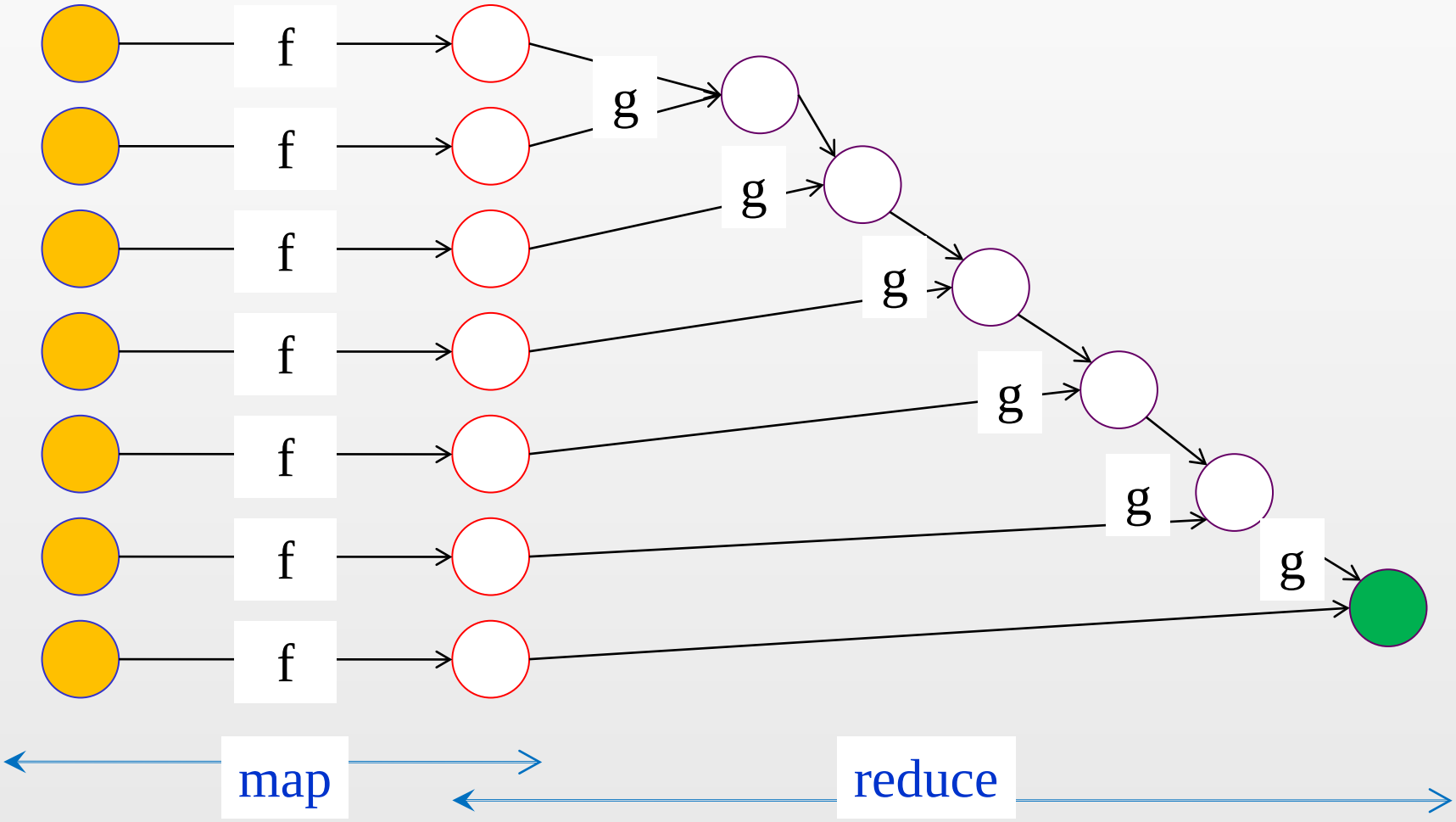
講義の内容

- MapReduce基礎
- (環境のセットアップ)
- 分散環境Hadoop
- HadoopにおけるMapReduce
- 演習
- 課題



MapReduce基礎

mapとreduce



map関数

- 入力
  - 関数 f（入力A, 出力B）
  - 型Aのオブジェクトのコレクション c
- 出力
  - 型Bのオブジェクトのコレクション
  - cの各要素に f を適用した結果の集まり

例題

- 例1
  - f : double → int, f(x) = floor(x + 0.5)
  - a = [1.2, 3.62, 7.8]
  - map(f, a) = [1, 4, 8]
- 例2
  - a = ["I", "am", "an", "engineer"]
  - map(length, a) = [1, 2, 2, 8]
  - map(upcase, a) = ["I", "AM", "AN", "ENGINEER"]

reduce（fold）関数

- 入力
  - 型Aのオブジェクトのコレクション c
  - 関数 f（入力A×A, 出力A）
- 出力
  - 型Aのオブジェクト
  - cの要素に f を次々と適用した結果

例題

- a = [5, 7, 1, 2]
- f : int × int ⟶ int , f(x, y) = x + y
- reduce(f, a) = 5 + 7 + 1 + 2 = 15
- g : int × int ⟶ int, g(x, y) = x * y
- reduce(g, a) = 5 * 7 * 1 * 2 = 70
- h : int × int ⟶ int, g(x, y) = x − y
- reduce(h, a) = 5 − 7 − 1 − 2 = −5 ??

  ((5 − 7) − 1) − 2 ≠ 5 − (7 − (1 − 2))

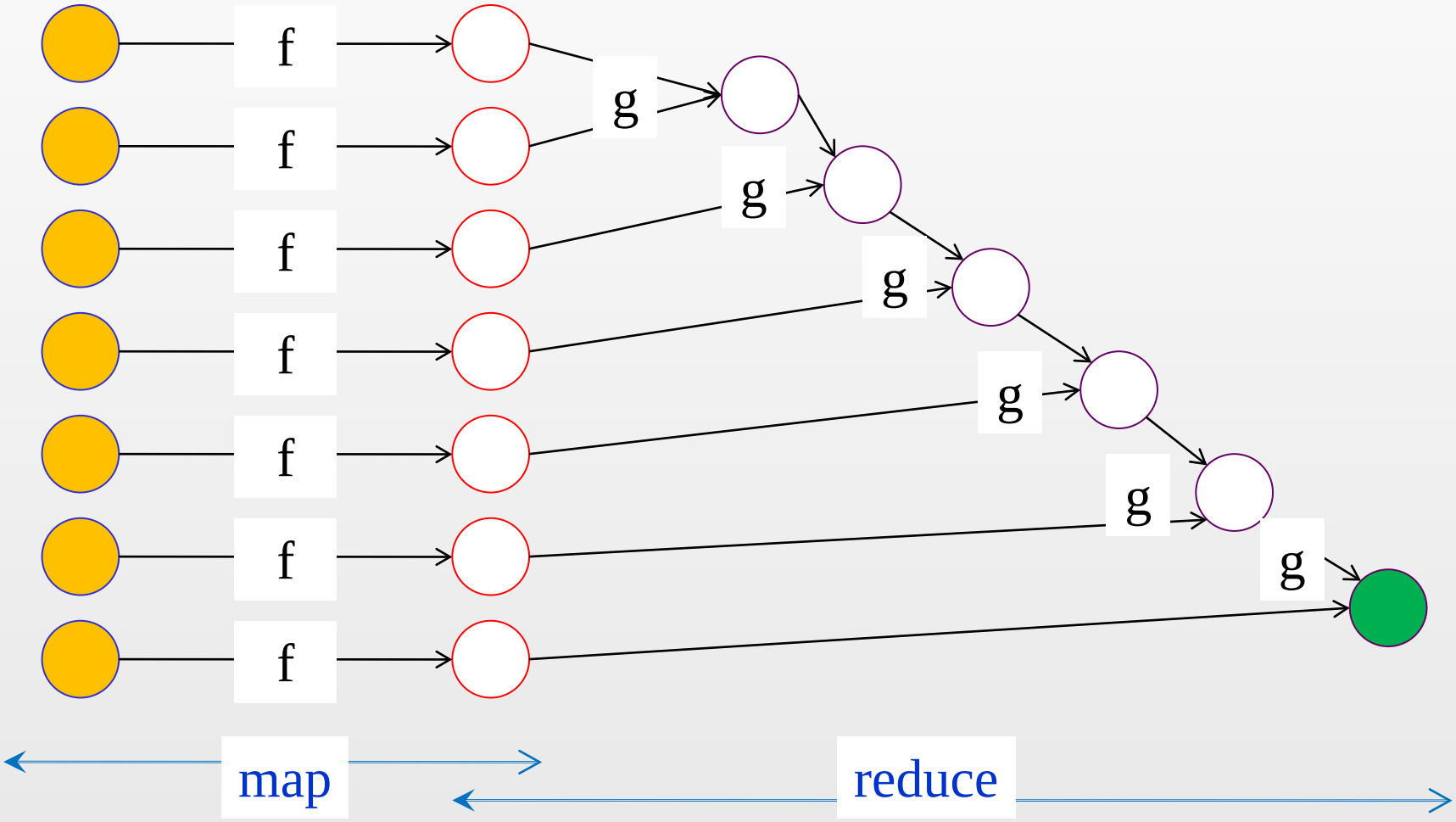
結合法則: f(a, f(b, c)) = f(f(a, b), c) を満たす関数を使う

map + reduce

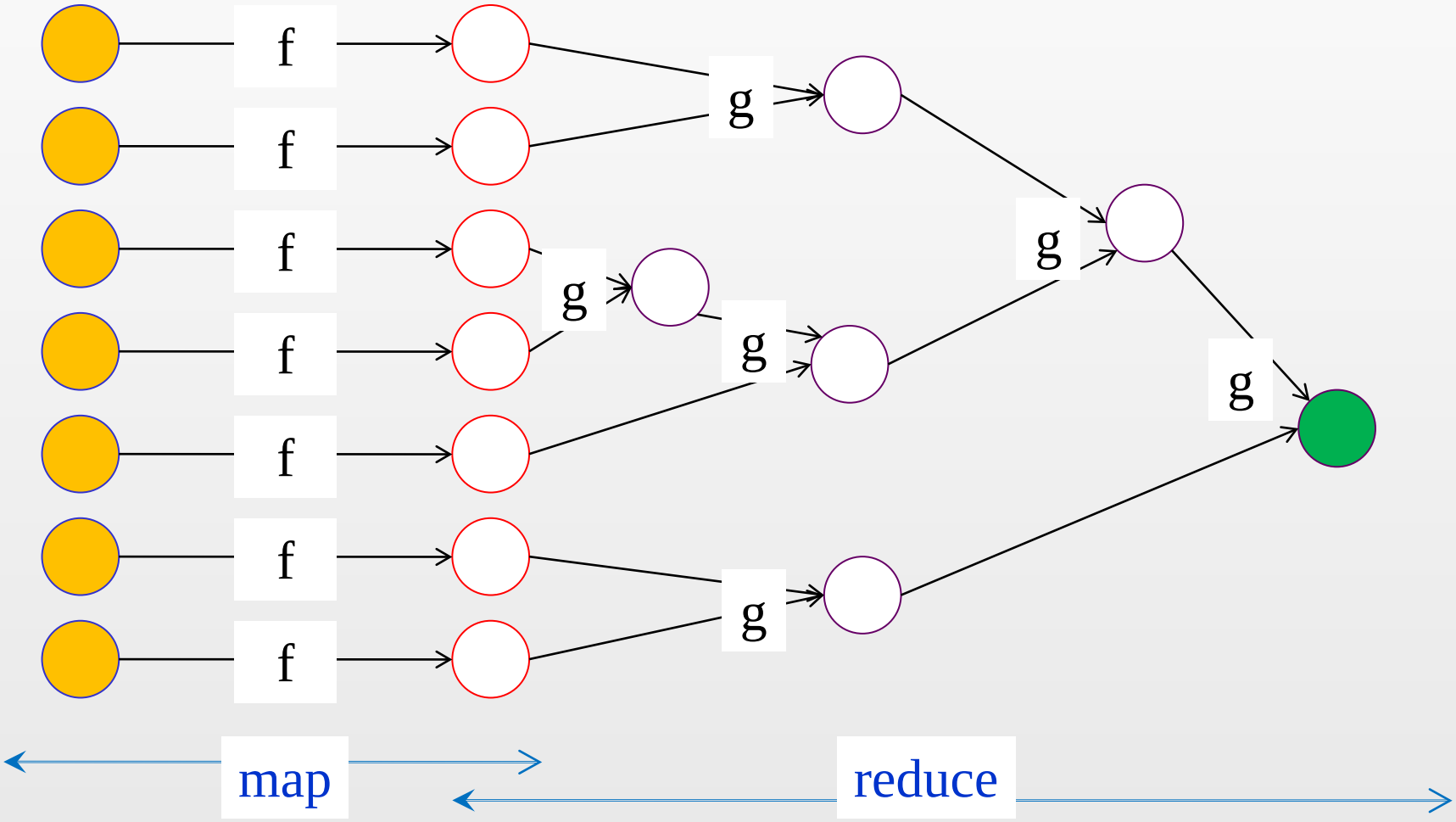
例題: 文字列のコレクションが与えられた時，長さの総和を求める.

- 入力: a = [″hello″, ″bye″, ...]
- b = map(length, a) = [5, 3, ...]
- totalLen = reduce(add, b) = 5 + 3 + ...
  - ここで，add(x, y) = x + y

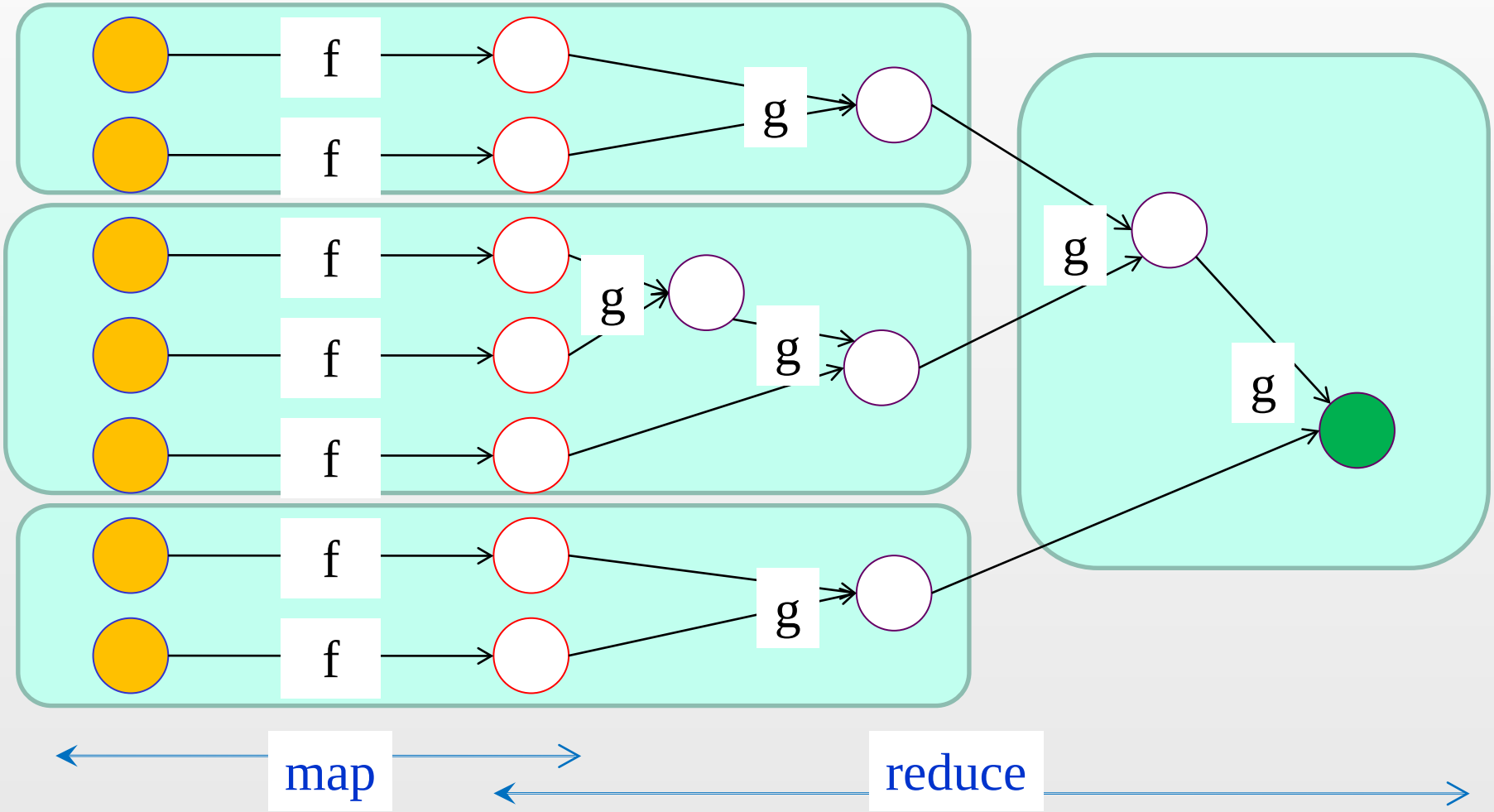
map, reduce と分散処理



map, reduce と分散処理



map, reduce と分散処理



map + reduce （再掲）

例題: 文字列のコレクションが与えられた時，長さの平均を求める.

- 入力: a = [″hello″, ″bye″, ...]
- b = map(length, a)
- totalLen = reduce(add, b)
- numStrings = length(a)
- mean = totalLen / numStrings

- a が巨大なコレクションで，length(a) が，簡単に計算できないかもしれない.

map + reduce

例題: 文字列のコレクションが与えられた時，長さの平均を求める.

- 入力: a = [″hello″, ″bye″, ...]
- b = map(length, a)
- totalLen = reduce(add, b)
- c = map(one, a) = [1, 1, ...]
  - ここで，one(x) = 1
- numStrings = reduce(sum, c)
- mean = totalLen / numStrings

クラウド環境のセットアップ

クラウドクライアントの設定確認

- 前回(先々週)と同じ設定を使う.

#	クラウドURL	プロジェクトチームID
4	https://vclc0004.ecloud.nii.ac.jp:8773/services/RDHC	h24-1-cld-04
6	https://vclc0006.ecloud.nii.ac.jp:8773/services/RDHC	h24-1-cld-06

- 前回出席していない方は，#6 を使う.

実習環境のセットアップ

- 以下のように，仮想マシンを3つ起動する.

項目	値
仮想マシンイメージ	mapred-2_EBS (misc)
インスタンスタイプ	m1.large
キーペア	(自分のものを使用)
インスタンス数	3
セキュリティグループ	mapred

- このうち1つを master に，残りの2つを slave にすることになるので，決めておく.

実習環境のセットアップ（つづき）

- クラウドクライアント上で，自分のキーペアが設定されている3つのインスタンスが起動したことを確認する．「名称」欄に，「mapred-2_EBS(vol-xxxxxxxx)」と表示されている．
- きわめて重要: クラウドクライアント右上で，「ボリューム・スナップショット」タブを選択し，ボリュームID欄で，上の vol-xxxxxxxx に対応する3行を見つける．各行上で右クリックし，「説明の編集」を選び，現れたウィンドウ上で，自分の名前と，master/slaveの別を記入する．
- 念のため，3つの vol-xxxxxxxx を，メモしておく．
- 重要: 立ち上げた3つのインスタンスの プライベートIPアドレス をメモしておく．

サーバプログラムの起動

- クラウドクライアントの仮想マシン一覧リストのmasterにすると決めたインスタンスの行で，シェルを起動する．
- 次のように，スクリプトを実行する．

```
[root@master ~]# script/vminit
```

注意: このスクリプトを実行すると，分散ファイルシステムが初期化される．

- 途中でいくつか質問をされる．
  - master, slave1, slave2 のプライベートIPアドレスを答える．入力を間違うと他人に迷惑がかかる可能性があるので注意．
  - 新規ユーザの作成をするかと聞かれるので，Yesと答える．ユーザ名とパスワードを適当に決めて入力する．以後，このスライドではユーザ名のところを *topse* と書くので，自分が決めたユーザ名に読み替えること．
  - 最後に(3台の)再起動をするかどうか尋ねられるので，Yesを答える．
- いったんログアウトし，ログインできるようになるまでしばらく待つ．

vncサーバの起動を確認

- Internet Explorer から，次のURLにアクセスする．
  http://xxx.xxx.xxx.xxx:5801/
  ただし，xxx.xxx.xxx.xxx は，master のパブリックIPアドレス．
- ダイアログボックスが2回現れる．
  - 最初は，OKを押す．
  - 次は，*topse*のパスワードを入力する．
- 通常のX Window画面が現れる．
  - 端末を起動: アプリケーション→アクセサリ→Gnome端末と選ぶ．
  - eclipseを起動: 端末で eclipse と入力する．
    デスクトップにランチャを作っておくと便利．
    (/usr/local/bin/eclipse)

vncサーバについて

- vncの画面のサイズを変えたいときには，vncserverを一旦終了した後，画面のサイズを指定して起動する.
  - 注意: クラウドクライアントからシェルを起動して行うこと!
- 作成したユーザになる

```
[root@master ~]# su - topse
```

- vncserver の終了:

```
[topse@master ~]$ vncserver -kill :1
```

ディスプレイ番号

- vncserver の起動:

```
[topse@master ~]$ vncserver -geometry 1800x1100
```

画面サイズ

HDFS, MapReduce サーバの起動確認

- (vncの画面で) firefox を起動する.
- HDFSサーバの起動を確認するため，次のURLにアクセスする:
  `http://localhost:50070/`
  - ページが表示され，Cluster Summaryの表の中のLive Nodes の値が2になっていればよい.
- MapReduceサーバの起動を確認するため，次のURLにアクセスする:
  `http://localhost:50030/`
  - ページが表示され，Cluster Summary の表の中のNodes の値が2になっていればよい.

HDFS, MapReduceサーバについて

- HDFS (Hadoopの分散ファイルシステム) は，以下の3つのサーバで実現されている.
  - namenode: マスターで動作
  - datanode: スレーブで動作
  - secondarynamenode: 今回は不使用
- これらのサーバは，マスターでユーザhdfsが以下のコマンドを実行することですべて起動・停止できる.
  - 起動: /usr/lib/hadoop/bin/start-dfs.sh
  - 停止: /usr/lib/hadoop/bin/stop-dfs.sh
- MapReduceフレームワークは，以下の2つのサーバで実現されている.
  - jobtrackder: マスターで動作
  - tasktracker: スレーブで動作
- これらのサーバは，マスターでユーザhdfsが以下のコマンドを実行することですべて起動・停止できる.
  - 起動: /usr/lib/hadoop/bin/start-mapred.sh
  - 停止: /usr/lib/hadoop/bin/stop-mapred.sh

MapReduceアプリケーション実行環境確認

- 以下を実行．円周率の近似値が表示されればOK.

```
[topse@master ~]$ hadoop jar
     /usr/lib/hadoop/hadoop-examples.jar pi 4 10000
```

ソースファイル編集の準備

- LMS (http://lms.cm.ecloud.nii.ac.jp/) から，cld56.tar.gz をダウンロードする.
- vncの端末を使い，ホームディレクトリで展開

```
[topse@master ~]$ cd
[topse@master ~]$ tar xvfz cld56.tar.gz
```

ホームディレクトリ以外にダウンロードした場合は，フルパスで指定する.

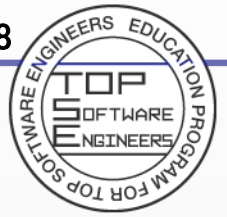
- eclipseを起動. workspaceは適当に指定.（初回なら）workbenchを選択する.
- File→Import→General→ Existing Projects into Workspace→Next
- Select root directoryで/home/*topse*/projを入力 →Finish

VMインスタンスについて

- 今回の講義(第5-6回)で用いるのは，「EBSインスタンス」と呼ばれるものである．前回使用したインスタンスとは異なり，シャットダウンしてもディスクの内容を消えないようにすることができる．
- クラウドクライアントで右クリックしたときに出るメニューの「インスタンス停止」にサブメニューがある．ここで「一覧に残す」を選択すると，シャットダウンは行われるがディスクの内容は消えない．
- 操作ミスの可能性は常にあるので，インスタンス上で作成した重要なファイルは小まめにローカルマシンにバックアップした方が良い．

ディスクサイズについての注意

- マスターサーバの，通常のディレクトリには，余り空き容量がない．1GBほど．
- ディレクトリ /mnt には，20GB弱の容量がある．ただし，ここは分散ファイルシステムの格納にも使用している．



分散処理基盤 Hadoop

Apache Hadoop

- Hadoop: 分散計算を実現するソフトウェア群
- サブプロジェクト
  - Common: 共通
  - HDFS: 分散ファイルシステム
  - MapReduce: 分散計算フレームワーク
- 関連プロジェクト
  - Avro: データシリアライズ
  - Cassandra: 多重データベース
  - Chukwa: データ収集・解析システム
  - HBase: 分散データベース
  - Hive: データウェアハウス
  - Mahout: 機械学習とデータマイニング
  - Pig: データフロー記述言語と実行系
  - ZooKeeper: 分散協調システム

Apache Hadoop

- オープンソース
- コモディティマシンで構成される
  クラスタシステムを想定
  - 故障は起こりうる前提
    → 多重化による信頼性確保
- 主たる利用言語: Java
  - Hadoop自身，Javaで構築されている
  - 他の言語もサポート

HDFS

- 分散ファイルシステム (Hadoop Distributed File System)
- 適する状況
  - 大きなファイル
  - 一度きりの書き込み，複数回の読み出し.
  - ストリーミング型のアクセス
  - コモディティハードウェア
- 不適な状況
  - 低レイテンシ要求
  - 大量の小さなファイル
  - 同一ファイルへの複数の書き込み者
  - ファイルの任意の位置で更新

HDFS の基本操作

- 書式: `hadoop fs` ***command***

- よく使われるコマンド
  - -help: ヘルプ
  - -ls <パス>: ファイル，ディレクトリの表示
  - -rm <ファイル>: 削除
  - -rmr <ディレクトリ>: 再帰的に削除
  - -put <コピー元>... <コピー先>: ローカルからコピー
  - -get <コピー元>... <コピー先>: ローカルにコピー
  - -cp <コピー元> <コピー先>: コピー
  - -mv <移動元> <移動先>: 移動

練習

- 適当な作業ディレクトリを作成し，そこに移る．何かファイルを作っておく．

```
$ mkdir /tmp/work
$ cd /tmp/work
$ echo abc > file1.txt
```

練習

- 以下のようなコマンドを実行してみよ．予想される出力が得られるか?

```
$ hadoop fs -lsr /
$ hadoop fs -mkdir /user/topse/dir1
$ hadoop fs -ls /user/topse
$ hadoop fs -copyFromLocal
                    file1.txt /user/topse/dir1/file2.txt
$ hadoop fs -lsr /user/topse
$ hadoop fs -copyToLocal
                    /user/topse/dir1/file2.txt file3.txt
$ ls -l
$ diff file1.txt file3.txt
$ hadoop fs -rmr /user/topse/dir1
$ hadoop fs -ls /user/topse
```

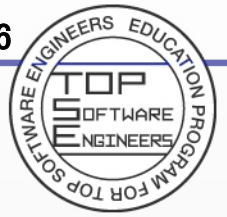
- hadoop fs −help の結果を見て，さらに適当に操作してみよ

注意事項

- 本来は，HDFSのファイルは以下のように指定する．

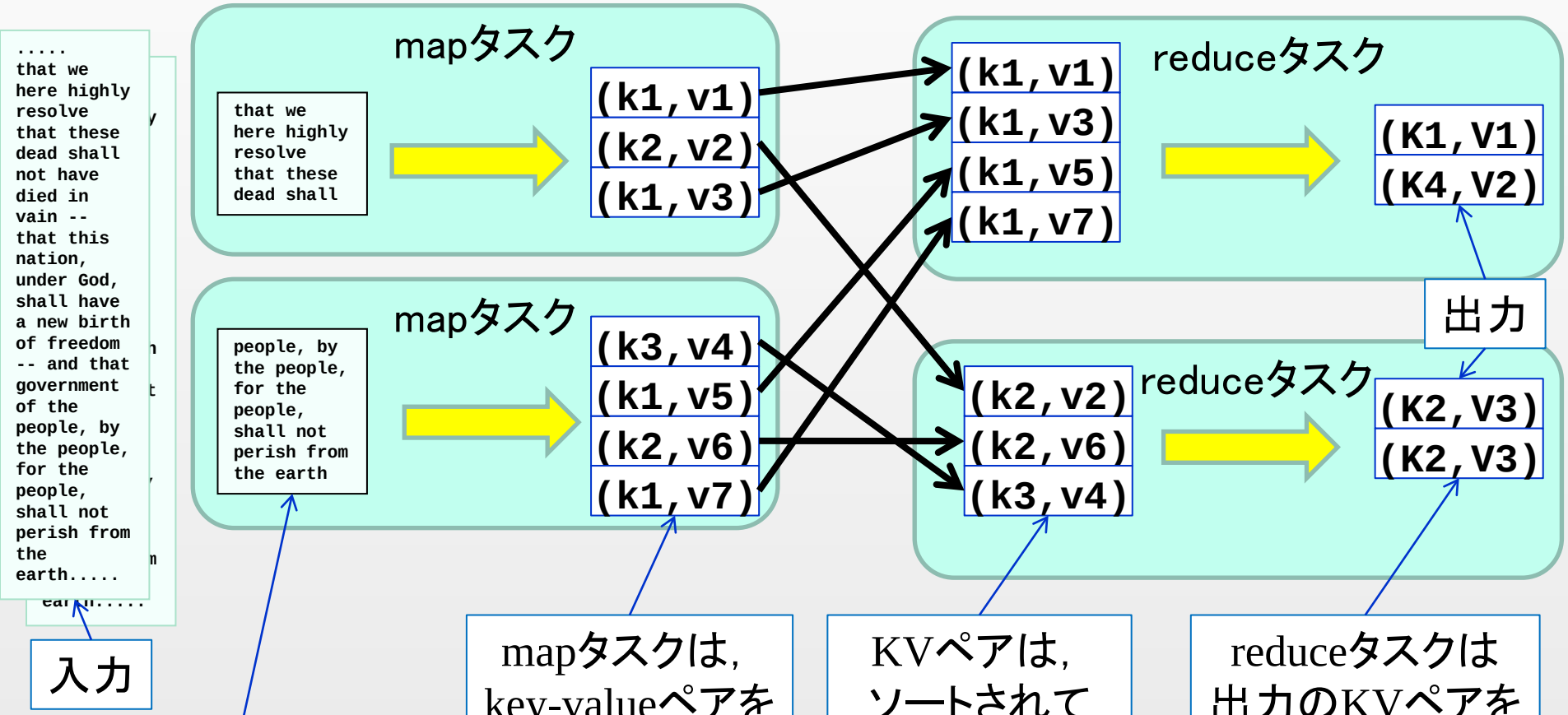
```
hdfs://masterserver:portno/thepath
```

- しかし，デフォルトとして
  hdfs://masterserver:portno
  の部分が設定されているので，/thepath の部分だけ記述すればよい．



HadoopにおけるMapReduce

HadoopのMapReduceジョブ概要



この全体をジョブと呼ぶ.

mapタスク

- Mapperクラス(の拡張クラス)が担当
- 前処理: setup メソッド
- 本体: map メソッド
  - 1行単位に分割したデータが渡される．すなわち，行数回 map メソッドが繰り返し呼び出される．
  - キーバリューペアを (0個以上) 作成して出力する．
- 後処理: cleanup メソッド

テキストデータの場合．
より正確には，
TextInputFormatクラスが
入力を処理する場合

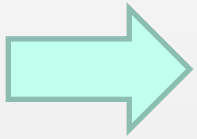
reduceタスク

- Reducerクラス(の拡張クラス)が担当
- 前処理: setupメソッド
- 本体: reduceメソッド
  - 同じキーのデータがすべてまとめられて，1回のreduceメソッド呼出に (Iterableとして) 渡される.
  - 各reduceタスクの中では，reduceの呼出は，キーでソートされて行われる.
- 後処理: cleanupメソッド

例題: WordCount

- 入力中に含まれる単語の一覧とその出現回数を出力する.

```
government of the
people, by the people,
for the people
```



```
by      1
for     1
government    1
of      1
people        3
the     3
```

WordCountでの map と reduce

- mapメソッド
  - 与えられた行の中での各単語wの出現に対し，KVペア（w, 1）を生成.
- reduceタスク
  - 各キー w に対し，渡されたペア (w, 1) の数 c を数え，KVペア（w, c) を生成.

WordCountでの map と reduce

mapタスクへの入力

```
of the people, by the
people, for the people
```

1回目のmapメソッド呼出

```
of the people, by the
```

1回目のmapメソッド出力

```
(of,1)
(the,1)
(people,1)
(by,1)
(the,1)
```

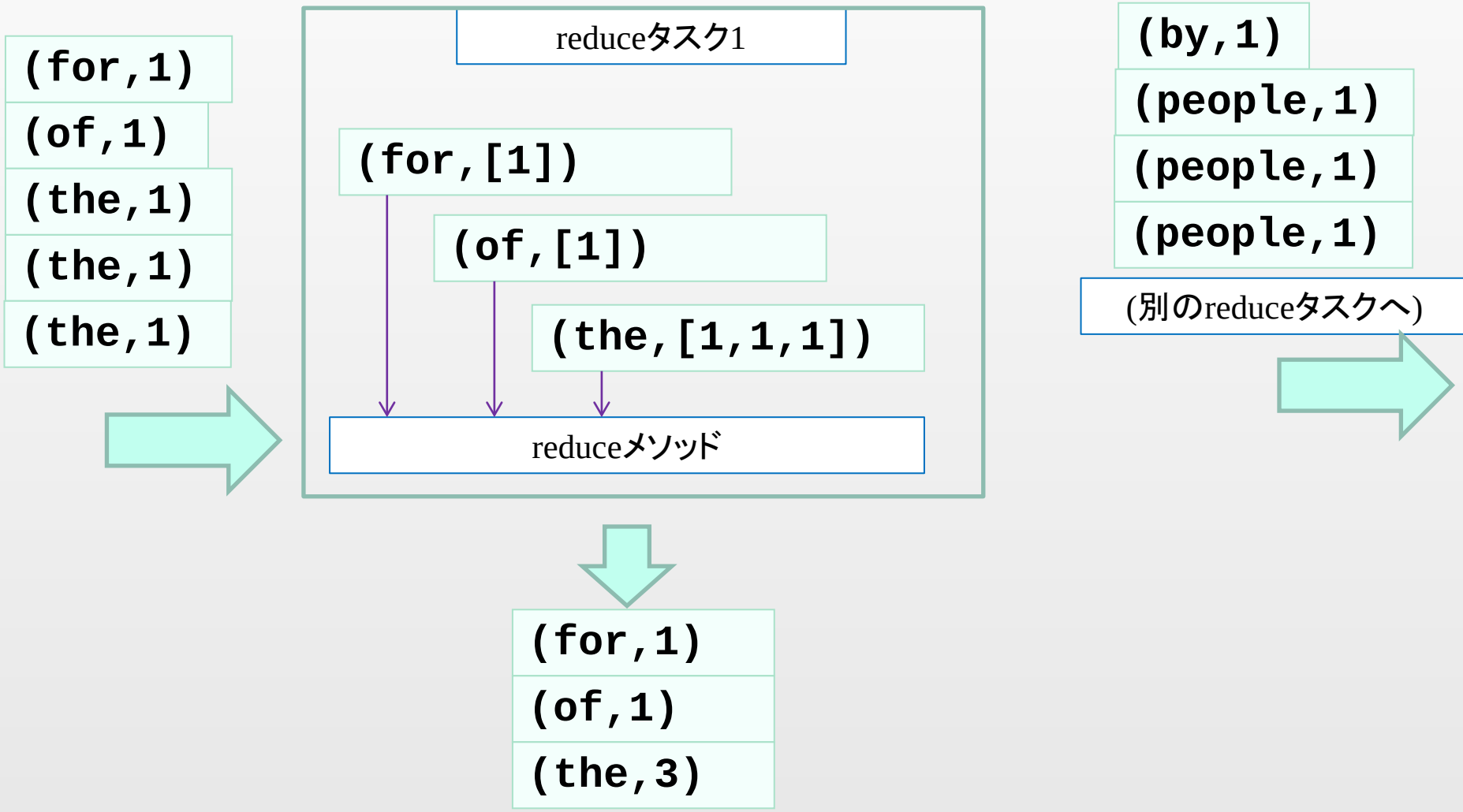
2回目のmapメソッド呼出

```
people, for the people
```

2回目のmapメソッド出力

```
(people,1)
(for,1)
(the,1)
(people,1)
```

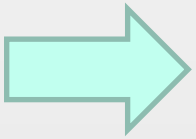
WordCountでの map と reduce



mapタスクへの入力

- ジョブの入力は，適当な KV ペアのコレクションとして，mapタスクに渡される．
  - KVペアの作り方はいろいろあり，対応するクラスがある．
  - TextInputFormatクラスの場合は，各行がバリュー．キーは，その行の，ファイル先頭からのバイトオフセット．

```
by the people,
for the people,
shall not perish from
```



(0, "by the people,")
(14, "for the people,")
(29, "shall not perish from")

Mapperクラス

- org.apache.hadoop.mapreduce.Mapperクラスを拡張したクラスのオブジェクトが，mapタスクを実現する．

```
public static class TokenizerMapper
    extends Mapper<Object, Text, Text, IntWritable> {

  public void map(Object key,
                  Text value,
                  Context context) throws ... { ...
  } ...
}
```

入力, キー　入力, 値　出力, キー　出力, 値

入力, キー　入力, 値

Wordcountの mapメソッド

(次スライド)

(次スライド)

```
private Text word = new Text();
private final static IntWritable one
                                  = new IntWritable(1);

public void map(Object key, Text value,
                Context context) throws ... {
  String line = value.toString();
  StringTokenizer tokenizer
                          = new StringTokenizer(line);
  while (tokenizer.hasMoreTokens()) {
    word.set(tokenizer.nextToken());
    context.write(word, one);
  }
}
```

WordCountでは無視

入力文字列

mapの出力

単語の切り出し

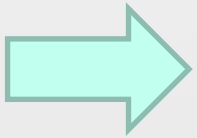
シリアライゼーション

- 整数，文字列などの値が，異なるマシン間で交換される．→ シリアライゼーションが必要
- シリアル化された整数: IntWritable クラス
  - int → IntWritable: コンストラクタ IntWritable(int i)
  - IntWritable → int: メソッド int get()
- シリアル化された文字列: Text クラス
  - String → Text: コンストラクタ Text(String s)
  - Text → String: メソッド String toString()

reduceタスク, reduceメソッド

- mapタスクの出力が，reduceタスクに渡される.
- 同じキーのデータは，必ず同じreduceタスクに渡される.
- キーごとに，reduceメソッドが呼び出され，対応する値が Iterable として渡される.
- reduceメソッドの呼出順は，キーでソートされる.

(k2,v7)
(k2,v6)
(k3,v4)



```
reduce(k2, [v7, v6]);
reduce(k3, [v4]);
```

Reducerクラス

- org.apache.hadoop.mapreduce.Reducerクラスを拡張したクラスのオブジェクトが，reduceタスクを実現する．

```
public static class IntSumReducer extends
        Reducer<Text, IntWritable, Text, IntWritable> {
  public void reduce(Text key,
    Iterable<IntWritable> values,
                  Context context) throws ... { ...
  } ...
}
```

入力, キー　入力, 値　出力, キー　出力, 値

入力, キー

入力, 値のコレクション

Wordcount の reduceメソッド

```
public void reduce(Text key,
  Iterable<IntWritable> values,
                  Context context) throws ... {
  int sum = 0;
  for (IntWritable v : values) {
    sum += v.get();
  }
  context.write(key, new IntWritable(sum));
}
```

入力の値を次々にvに入れて処理

IntWritable から Intに変換

reduceの出力

main

設定情報

設定情報をコマンドラインから追加

それ以外の引数

```
public static void main(String[] args) ... {
  Configuration conf = new Configuration();
  String[] otherArgs =
         new GenericOptionsParser(conf, args).
                                      getRemainingArgs();
  Job job = new Job(conf, "wordcount");

  FileInputFormat.setInputPaths(job,
                                new Path(otherArgs[0]));
  FileOutputFormat.setOutputPath(job,
                                new Path(otherArgs[1]));
```

入出力パスの指定

main

```
  job.setMapperClass(WCMapper.class);
  job.setReducerClass(WCReducer.class);

  job.setInputFormatClass(TextInputFormat.class);
  job.setOutputFormatClass(TextOutputFormat.class);
  job.setOutputKeyClass(Text.class);
  job.setOutputValueClass(IntWritable.class);
  job.setMapOutputKeyClass(Text.class);
  job.setMapOutputValueClass(IntWritable.class);
  boolean ok = job.waitForCompletion(true);
  System.exit(ok ? 0 : 1);
}
```

入力をMapperに渡すクラス

Reducerから受け取ったデータを出力するクラス

jobの実行

Mapper出力のKeyのクラス
(Reducerと同じ場合には省略可)

Reducer出力のKeyのクラス

Mapper出力のValueのクラス
(Reducerと同じ場合には省略可)

Reducer出力のValueのクラス

コンパイルと実行

- eclipseでは，コンパイルは自動的に行われ，classファイルが生成される．
- jarファイルを作成する．
  - eclipseのExport で生成するか，もしくは，
  - 端末を用い mkjar コマンドを実行すると，cld56.jar ファイルが proj ディレクトリに作成される．

```
$ cd ~/proj
$ script/mkjar
```

- MapReduceアプリケーションの実行書式:

```
hadoop jar jarfile classname args...
```

3つの実行モード

- Hadoopには，3つの実行モードがある:

モード	マシン台数	ファイルシステム
ローカル	1	ローカル
疑似分散	1	HDFS
完全分散	複数	HDFS

- この講義では，ローカルモードと完全分散モードを用いる.
  - 開発中は，まずローカルモードを用いる．いきなり完全分散モードでは，デバッグが困難.
  - ローカルモードではスケールしない．運用時の大量データは完全分散モードで処理.

実行モードの切替

- スーパーユーザになって，以下のように実行.

```
$ sudo su -
password for topse:
# update-alternatives --config hadoop-0.20-conf
There are 3 programs which provide 'hadoop-0.20-conf'.

  Selection    Command
-----------------------------------------------
   1           /etc/hadoop-0.20/conf.empty
   2           /etc/hadoop-0.20/conf.pseudo
*+ 3           /etc/hadoop-0.20/conf.distr

Enter to keep the current selection[+], or type selection number:
```

- 1を選択するとローカルモード，
  3を選択すると完全分散モードになる.

WordCountの実行

- WordCount は，2つの引数を取る.
  - 第1引数は，入力データを格納するディレクトリ（このディレクトリにあるすべてのファイルが処理される.
  - 第2引数は，出力データを格納するディレクトリ．コマンド起動時にこのディレクトリが存在してはいけない.
- ローカルモードでの実行例:

```
$ cd ~/proj
$ rm -rf output/wc
$ hadoop jar cld56.jar cld56.WordCount
                                    input/wc output/wc
```

WordCountの実行

- 完全分散モードでの実行方法

```
$ cd ~/proj
$ hadoop fs -mkdir /user/topse/input/wc/small
$ hadoop fs -put input/wc/*.txt
                              /user/topse/input/wc/small
$ hadoop fs -rmr /user/topse/output/wc
$ hadoop jar cld56.jar cld56.WordCount
   /user/topse/input/wc/small /user/topse/output/wc
```

- 結果の確認: ローカルファイルにコピーし，エディタで開く.

```
$ hadoop fs -get '/user/topse/output/wc/*'
                                          some/directory
```

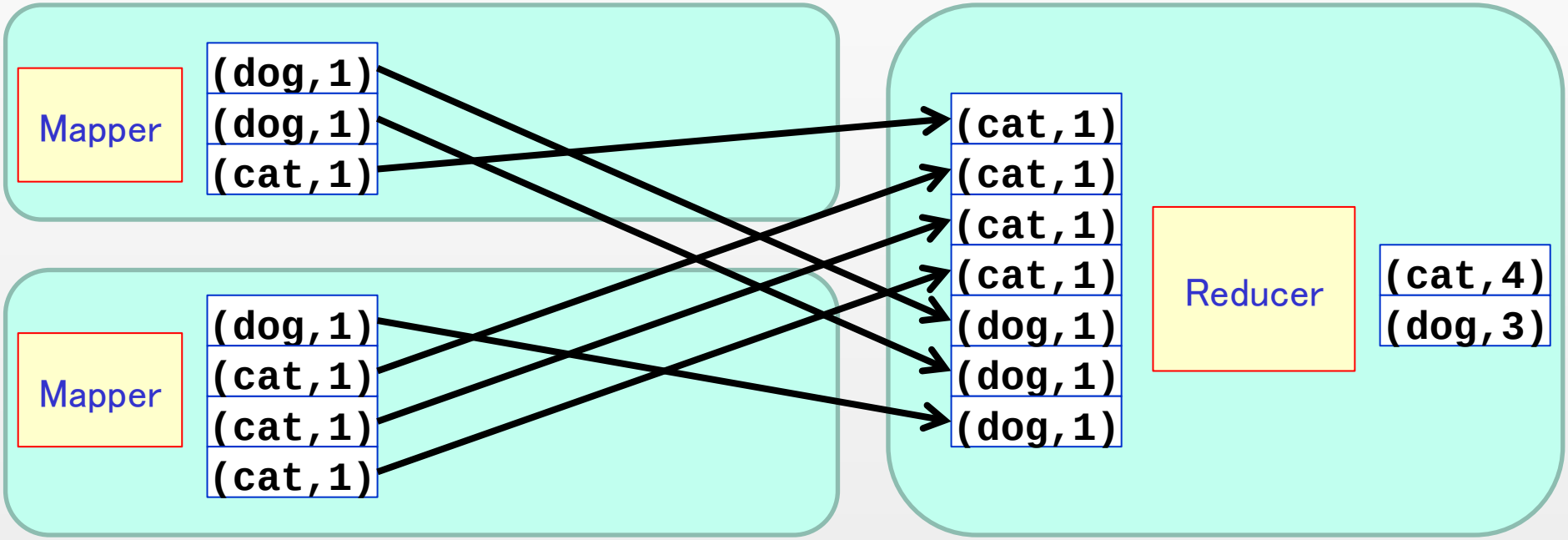
WordCountの実行（大きめのファイル）

- 小さなファイルでは，完全分散モードではオーバーヘッドが大きかった．
- proj/script/mkd-wc を実行すると，/mnt/indata/wc に大きめのファイルがいくつか作成される．（数分かかる）
- 完全分散モード，ローカルモード等で実験して性能を比較してみよ．
- mkd-wcはスクリプトなので，興味がある場合にはサイズを変えて実験してみると良い．

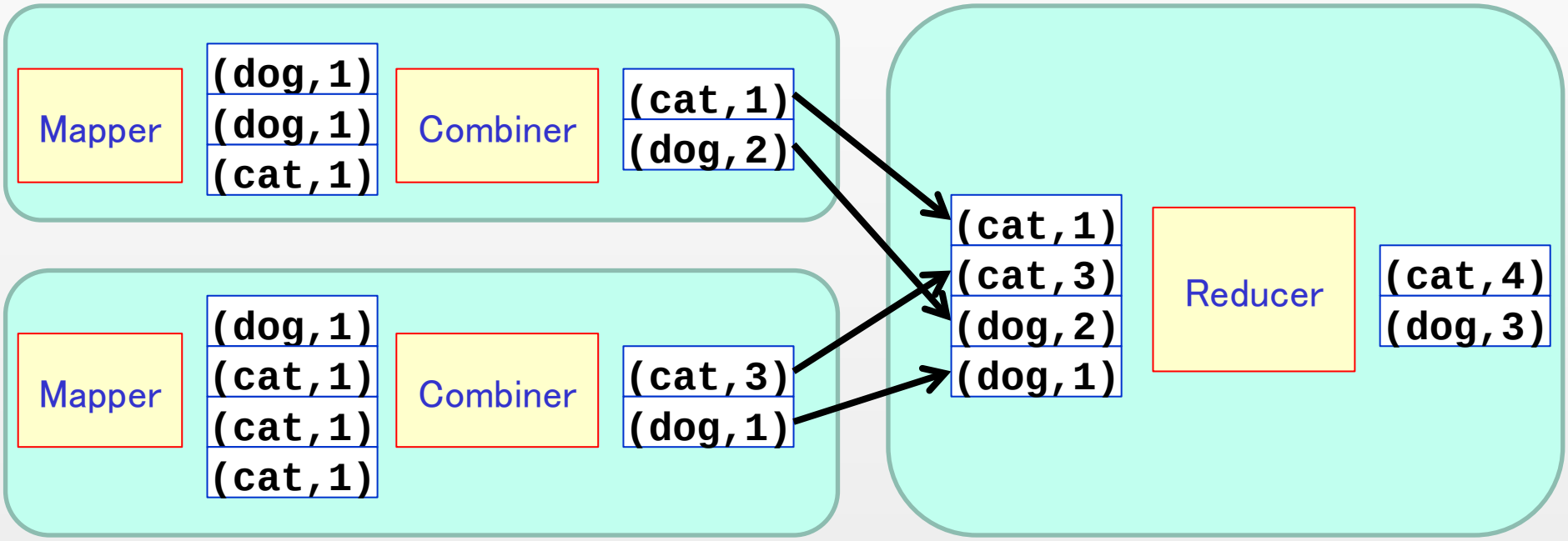
combiner

- mapperの出力が，reducerにわたる時には，ネットワーク経由となる．データ量を減らしたい．
- combiner
  - 入力: mapperの出力と同じ型のKVペア
  - 出力: reducerの入力と同じ型のKVペア
  - つまり，入力と出力は同じ型
- mapperの出力が，reducerに渡されるまでの間に，0回以上呼ばれる．
  - 典型的には，mapperの出力がローカルで処理される．

combiner（つづき）



combiner（つづき）



- wordcount では，reducerと全く同じcombinerが使える．
- いつでも そうできる というわけではない．

combiner（つづき）

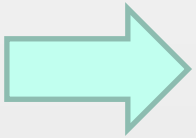
- combinerも，Reducerクラスを拡張して作成する．（ただし，入力KVと出力KVは同じ型．）
- mapper, reducerと同様に，Job クラスの setCombinerClass() を使って，指定を行う．
  - WordCount.java 中で，コメントアウトされている．

演習

演習1

オークションサイトにおける，出品者と，それに対する応札者のリストが与えられている．各参加者ごとに，その人が応札したことのある出品者のリストを作りたい．たとえば，左の入力に対し，右が出力となる．

```
usr1  usr3,usr6,usr2
usr7  usr3
usr1  usr6,usr7
```



```
usr2  usr1
usr3  usr7,usr1
usr6  usr1
usr7  usr1
```

演習1（つづき）

- Exc1.java は，作成中のソースコードである．これを完成させよ．
- combinerについても考慮せよ．
- 小さな入力データが，proj/input/exc1 に用意されている．スクリプト mkd-exc1を実行することで，大きめのデータが/mnt/indata/exc1に作成される．スクリプトを書き換えることで大きさを調整することができる．

演習2

ある電子店舗の取引を記録したファイル群がある．各行はタブで区切られており購入者，商品ID，数量が記録されている．商品ごとの単価は別のファイル群に書かれており，同じフォーマットで商品IDと単価が記録されている．簡単のため，取引記録の各行の先頭のカラムには1が，単価表には2がマーカーとして入っているものとする．[ここでは，（価格改定ごとに新たなIDが付与されるなどの理由で）商品ID数が非常に多く，メモリには乗りきらないものと仮定する．]

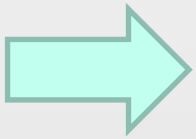
購入者ごとの合計購入金額を計算せよ．

商品ID　購入者　数量

```
1 idQm nameB 5
1 idPc nameA 7
1 idQd nameB 3
```

商品ID　単価

```
2 idPc 20
2 idQd 20
2 idQm 30
```



```
nameA 140
nameB 210
```

idQm ... 30*5 = 150
idQd ... 20*3 = 60
合計 210

演習2（つづき）

- 小さな入力データが，proj/input/exc2 に用意されている．スクリプト proj/script/mkd-exc2を実行することで，大きめのデータが /mnt/indata/exc2に作成される．スクリプトを書き換えることで大きさを調整することができる．

演習3

通信機器を持って移動しているユーザが多数いる．あるユーザaが別のユーザbに通信する場合，aの近隣にある基地局Pから，中央局を通り，さらにbの近隣にある基地局Qを通すことになる．簡単のため，通信は瞬時に終わるものとし，また，すべてのユーザは，いつでもちょうど1つの基地局の電波到達範囲にいるものとする．また，各ユーザは同時刻には2つ以上の通信を行わないものとする．

各基地局は，ユーザが電波到達範囲に入った時刻を記録している．

中央局には，2人のユーザ間の通信記録（2人のユーザ名と通信時刻）がある．

各ユーザごとの発信履歴を作成せよ．各エントリには，通信相手，通信時刻，発信局，受信局を含めること．

［ユーザ単位では，全情報がメモリに乗る程度だと仮定できるものとする．］

演習3（つづき）

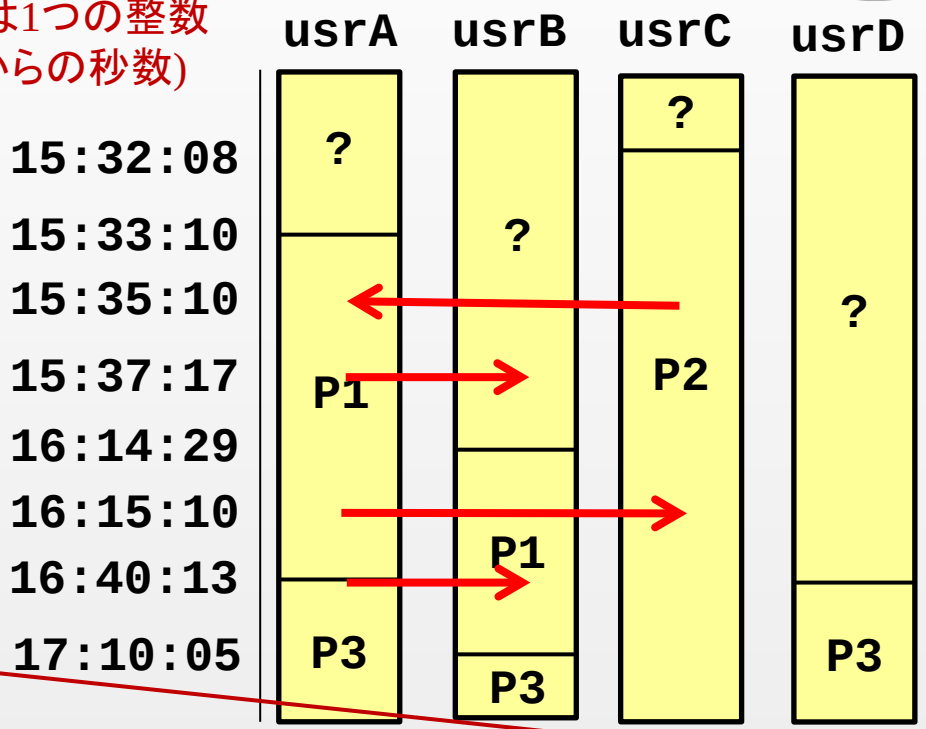
基地局ログ

```
P1 usrA 15:33:10
P1 usrB 17:10:05

P2 usrC 15:32:08

P3 usrB 16:14:29
P3 usrA 16:40:13
P3 usrD 16:40:13
```

実際には1つの整数(epochからの秒数)が入る



同時刻は，移動済みとみなす

発信者ごとにとりまとめ，時刻順にソート

出力

```
usrA 16:15:10 usrC P1 P2
usrA 16:40:13 usrB P3 P3
usrC 15:35:10 usrA P2 P1
```

通信ログ

```
C 15:35:10 usrC usrA
C 15:37:17 usrA usrB
C 16:15:10 usrA usrC
C 16:40:13 usrA usrB
```

一方の場所が不明の場合には出力しない

マーカー"C"，時刻，発信者，受信者

発信者，時刻(実際には整数)，受信者，発信局，受信局の順

演習3（つづき）

- 小さな入力データが，proj/input/exc3 に用意されている．スクリプト proj/script/mkd-exc3 を実行することで，大きめのデータが /mnt/indata/exc3 に作成される．（だいぶ時間がかかる．）スクリプトを書き換えることで大きさを調整することができる．

レポート課題

- 演習3の解答を作成してください．(実際の進捗状況によって課題を変更する可能性があります．講義での指示およびLMSの記述に注意してください．)
- 提出物: ソースコード，jarファイル，解答の方針/プログラムの説明．
- 上記のファイル群を tar.gz または zip で，1つのファイルにまとめて，LMSに提出してください．
- 提出期限: 2012/5/28 (月) 午前9時　厳守 (期限後の提出は原則として受け付けませんが，事情のある人は相談してください)
- 正しく動作する jar ファイルが提出された場合には，解答の方針やプログラムの説明は，ごく簡単で構いません (無くても良いです)．そうでない場合には，解答の方針やプログラムの説明に応じて部分点を出しますので，自信がない場合には詳細に説明してください．
- 提出者には，提出期限後に解答例を配布します．原則としてそれ以上のフィードバックはありません．疑問や質問のある方は，提出前・後にかかわらず，遠慮無く田辺にメール (cld@topse.jp) してください．(提出期限後でも構いません)
- 単位が必要な人は，期限までに必ず提出してください．正解にたどり着いていなくても，説明を十分書いてあれば良いです．

参考書・文献

- 太田, 下垣, 山下, 猿田, 藤井, 濱野: 「Hadoop徹底入門」(翔泳社), 2011.
- Tom White著, 玉川・兼田訳: 「Hadoop」(オライリー・ジャパン), 2010.
- Jimmy Lin, Chris Dyer: "Data-Intensive Text Processing with MapReduce" (Morgan & Claypool Publishers), 2010.
- Jeffrey Dean, Sanjay Ghemawat: MapReduce: Simplified Data Processing on Large Clusters. OSDI'04, pp.137-150, 2004.
- Apache Hadoopウェブページ: http://hadoop.apache.org/