インターネット計測とデータ解析 第 13 回

長 健二朗

2014 年 7 月 7 日

前回のおさらい

第 12 回 検索とランキング (6/30)

- 検索システム
- ページランク
- 演習: PageRank

今日のテーマ

第 13 回 スケールする計測と解析

- 大規模計測
- クラウド技術
- MapReduce
- 演習: MapReduce

計測、データ解析とスケーラビリティ

計測手法

- 測定マシン側の回線容量、データ量、処理能力

データ収集

- 複数箇所からデータを集める
- 収集マシン側の回線容量、データ量、処理能力

データ解析

- 膨大なデータの解析
- 比較的単純な処理の繰り返し
- データマイニング手法による複雑な処理
- データ解析マシン側のデータ量、処理能力、分散処理の場合は通信能力

計算量 (computational complexity)

アルゴリズムの効率性の評価尺度

- 時間計算量 (time complexity)
- 空間計算量 (space complexity)

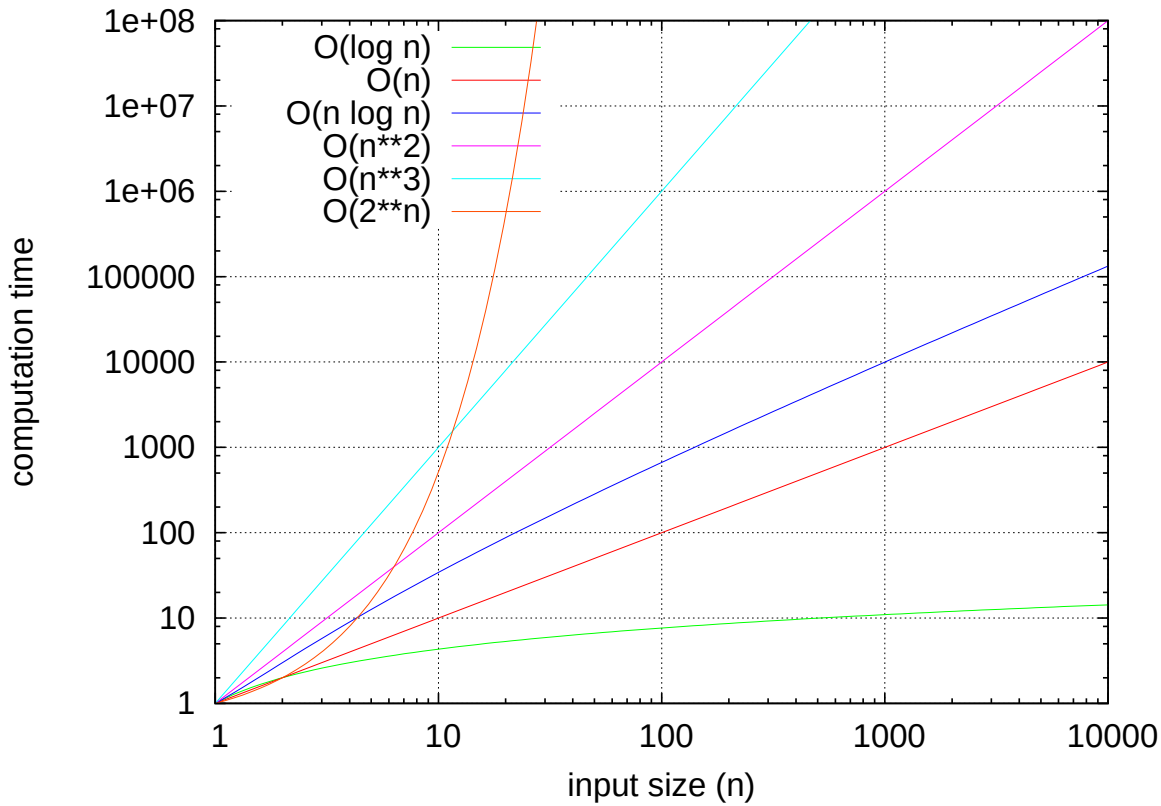
- 平均計算量
- 最悪計算量

オーダー表記

- 入力数 n の増大に対して計算量の増加する割合をその次数のみで表現
  - 例: $O(n), O(n^2), O(n \log n)$
- より正確には、「$f(n)$ はオーダー $g(n)$」とは、ある関数 $f(n)$ と関数 $g(n)$ に対して $f(n) = O(g(n)) \Leftrightarrow$ ある定数 C, n_0 が存在して、$|f(n)| \leq C|g(n)|\ (\forall n \geq n_0)$

時間計算量

- 対数時間 (logarithmic time)
- 多項式時間 (polynomial time)
- 指数時間 (exponential time)



計算量の例

サーチ

- リニアサーチ: $O(n)$
- バイナリサーチ: $O(\log_2 n)$

ソート

- 選択整列法: $O(n^2)$
- クイックソート: 平均で $O(n \log_2 n)$ 、最悪は $O(n^2)$

一般に、

- 全変数を調べる (ループ): $O(n)$
- バイナリツリー構造など: $O(\log n)$
- 変数に対する 2 重ループ: $O(n^2)$
- 変数に対する 3 重ループ: $O(n^3)$
- 全変数の組合せ (最短経路検索など): $O(c^n)$

分散アルゴリズム

並列アルゴリズム

- 問題を分割して並列実行
- 通信コスト、同期問題

分散アルゴリズム

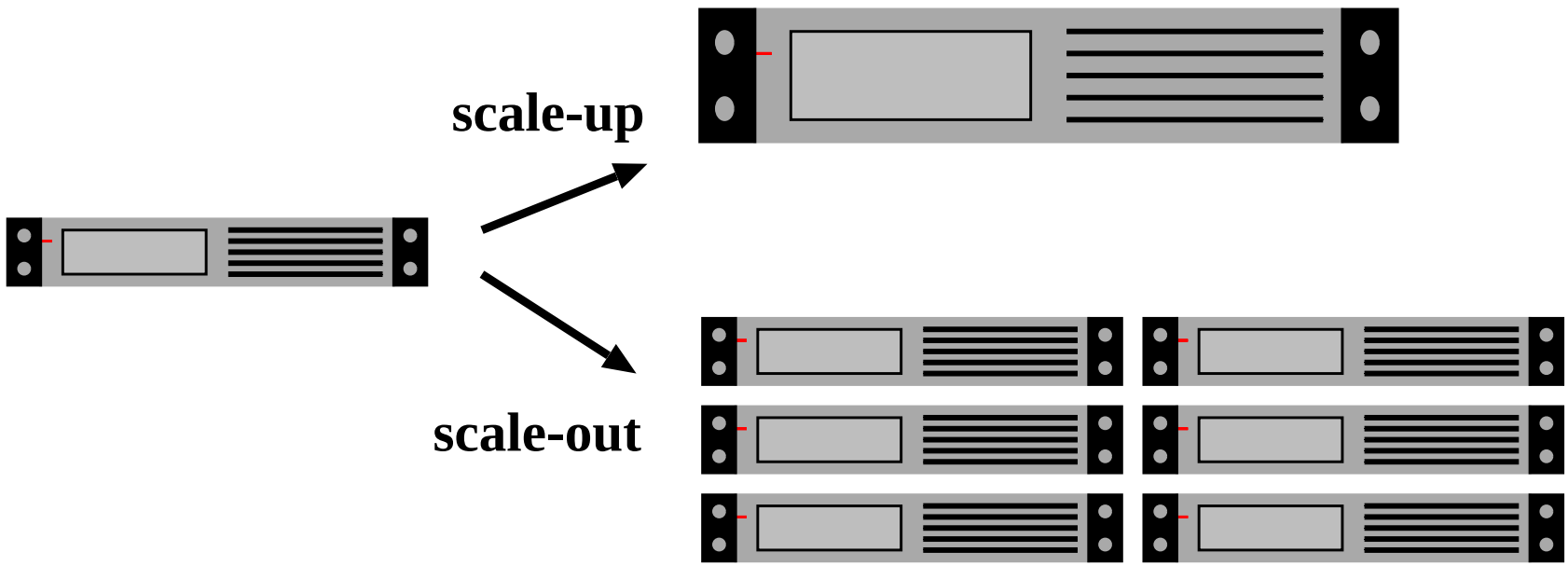
- 並列アルゴリズムのなかでも、独立したコンピュータ間のメッセージ交換のみによる通信を前提にしたもの
- コンピュータの故障やメッセージの損失を考慮

メリット

- スケーラビリティ
  - しかし、最善でも並列度に対してリニアな向上
- 耐故障性

スケールアップとスケールアウト

- スケールアップ
  - 単一ノードの拡張、強化
  - 並列処理の問題がない
- スケールアウト
  - ノード数を増やすことによる拡張
  - コスト効率、耐故障性 (安価な量産品を大量に使う)



クラウド技術

クラウド: さまざまな定義がある

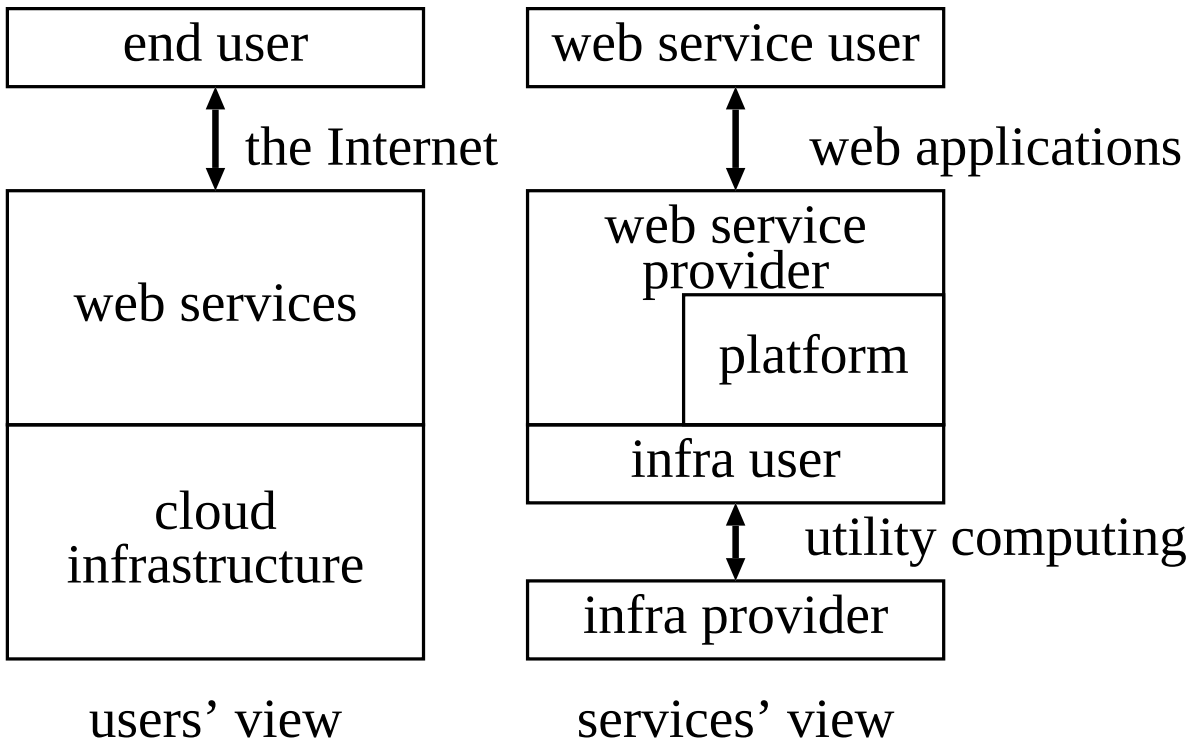
- 広くは、ネットワークの向うにあるコンピュータ資源

背景

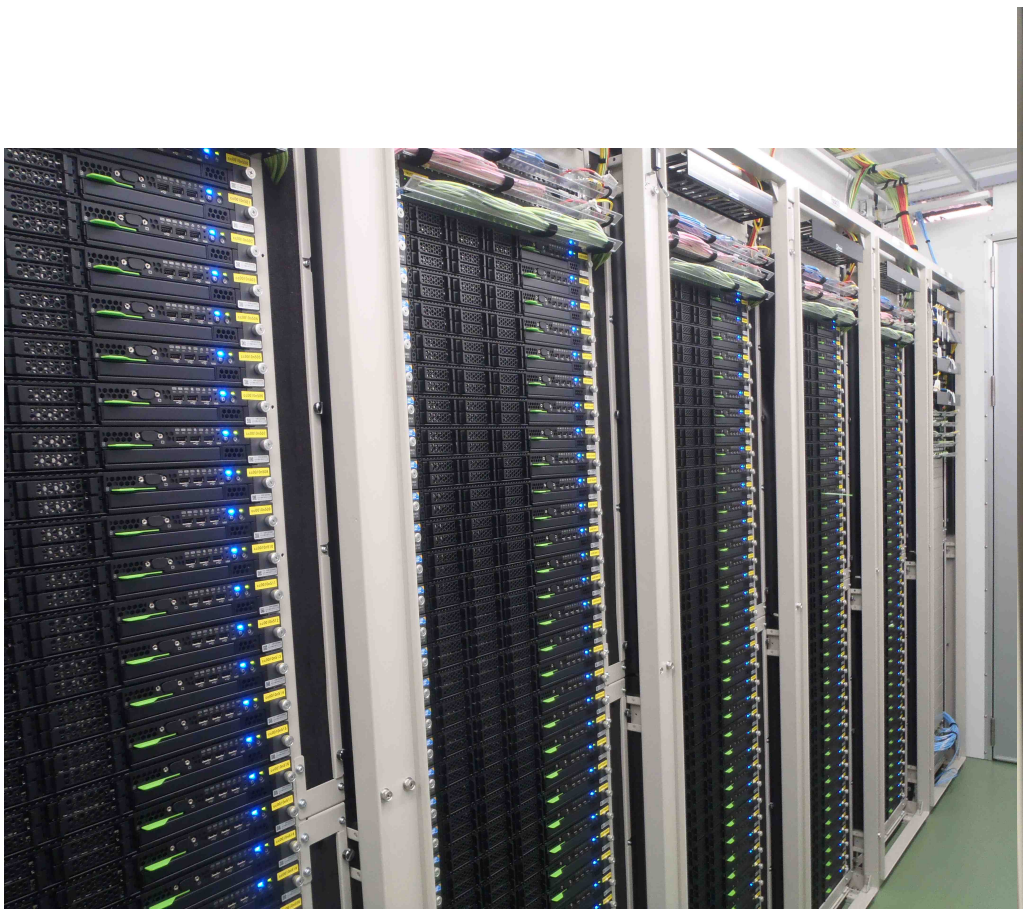
- 顧客ニーズ:
  - 計算資源、管理、サービスのアウトソース
  - 初期費用が不要、需要予測をしなくていい
  - それによるコスト削減
- 震災以降はリスク回避と省エネにも注目
- プロバイダ: スケールメリット、囲い込み

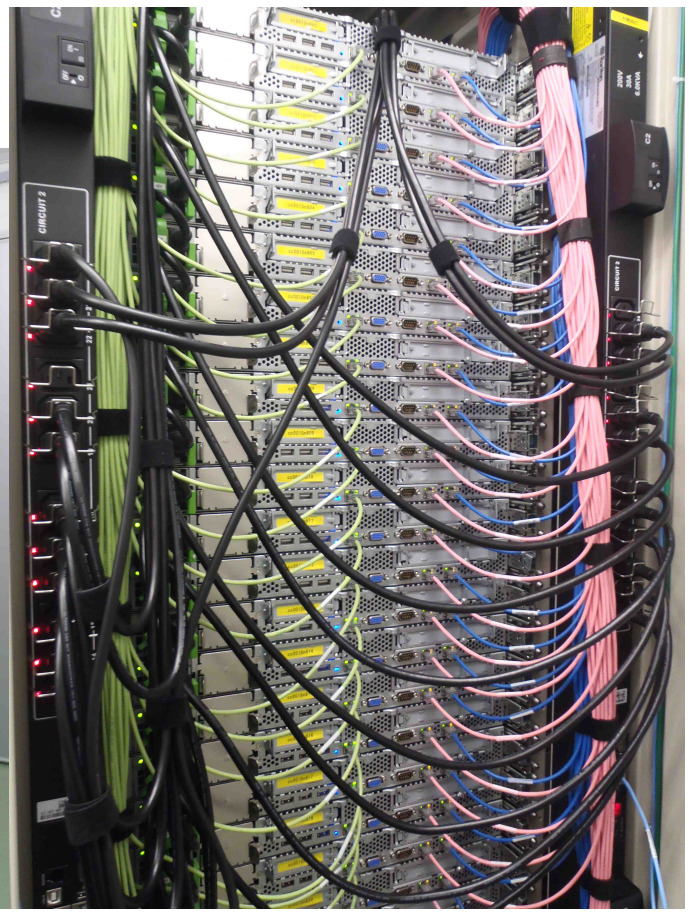
さまざまなクラウド

- public/private/hybrid
- サービス形態: SaaS/PaaS/IaaS

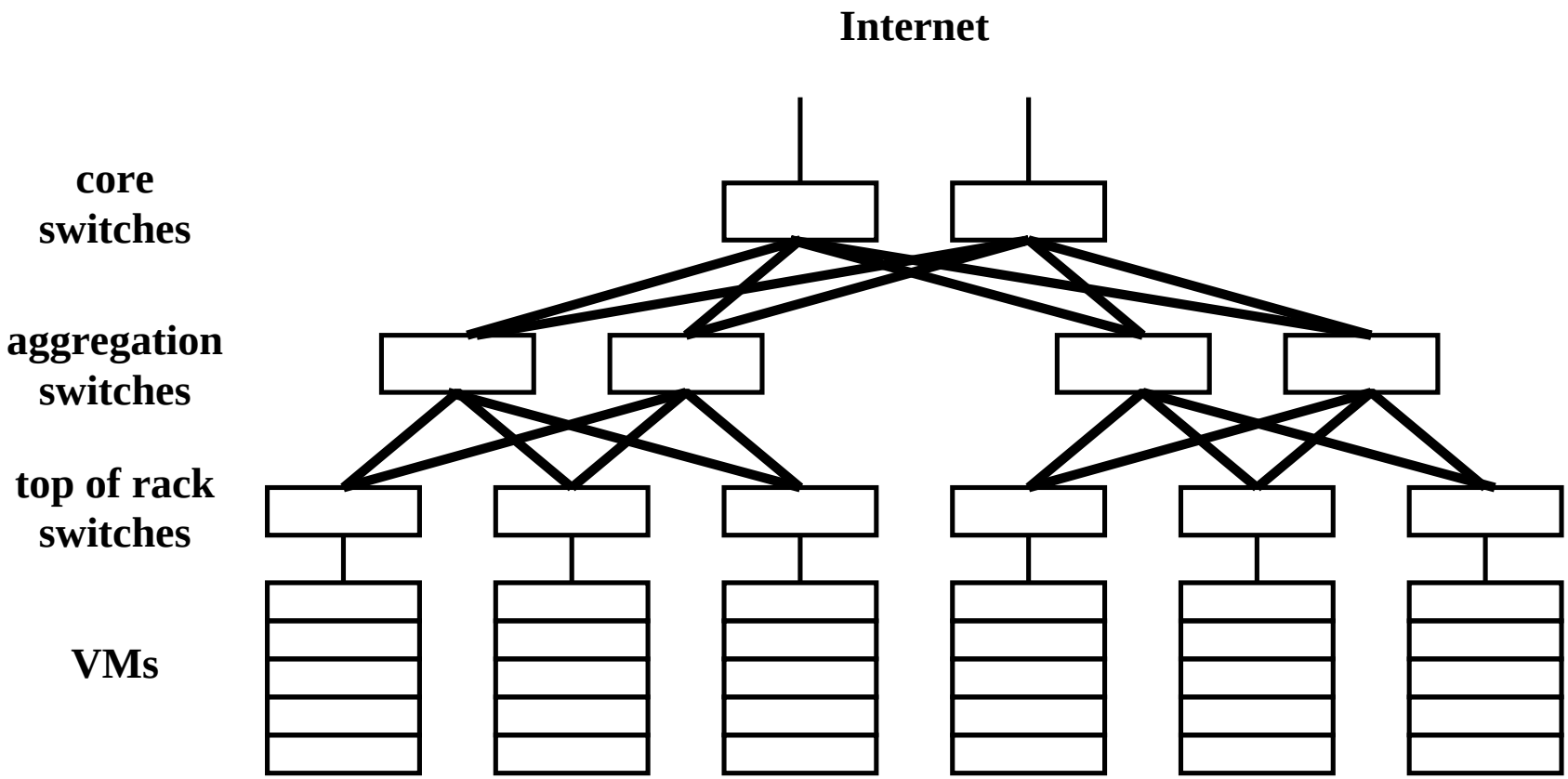


physical clouds





typical cloud network topology



キーテクノロジー

- 仮想化: OS レベル、I/O レベル、ネットワークレベル
- ユーティリティコンピューティング
- 省エネ、省電力、低発熱
- データセンターネットワーキング
- 管理、監視技術
- 自動スケーリング、ロードバランシング
- 大規模分散データ処理技術

- 関連研究分野: ネットワーク、OS、分散システム、データベース、グリッド
  - 結構商用サービスの方が先を行っている

クラウドの経済

- 規模の経済 (調達コスト、運用コスト、統計多重効果)
- コモディティハードウエア
- 廉価な場所 (含む空調、電気代、ネットワーク)

日本のクラウドはグローバルに戦えるのか？
(大きいことはいいことか？)

集約と分散の技術スパイラル

- タイムシェアリング - ワークステーション/PC - クラウドと thin clinet/NetPC
- スイッチ／ルータのポート数

- 技術が成熟してくると、規模の経済を求め集約へ
- 技術変化が起こると、柔軟性を求め分散へ

必ずしも大規模クラウドが生き残るとは限らない

さらなる技術革新の可能性！

MapReduce

MapReduce: Google が開発した並列プログラミングモデル

Dean, Jeff and Ghemawat, Sanjay.
MapReduce: Simplified Data Processing on Large Clusters.
OSDI'04. San Francisco, CA. December 2004.
`http://labs.google.com/papers/mapreduce.html`

本スライドの MapReduce 部分はこの資料から作成している

動機: 大規模データ処理

- 何百、何千台規模の CPU を利用してデータ処理したい
- ハードウェア構成等を意識せず簡単に利用したい

MapReduce のメリット

- 並列分散処理の自動化
- 耐故障性
- I/O スケジューリング
- 状態監視

MapReduce プログラミングモデル

Map/Reduce

- Lisp や他の関数型言語からのアイデア
- 汎用性: 幅広い応用が可能
- 分散処理に適している
- 故障時には再実行可能

Map/Reduce in Lisp
(map square '(1 2 3 4)) $\rightarrow$ (1 4 9 16)
(reduce + '(1 4 9 16)) $\rightarrow$ 30

Map/Reduce in MapReduce

map(in_key, in_value) → list(out_key, intermediate_value)

- key/value ペアのセットを入力に、別の key/value ペアを生成

reduce(out_key, list(intermediate_value)) → list(out_value)

- map() で生成された結果を使い、特定の key に対応する value をマージした結果を返す

例: 文書内の単語の出現頻度のカウント

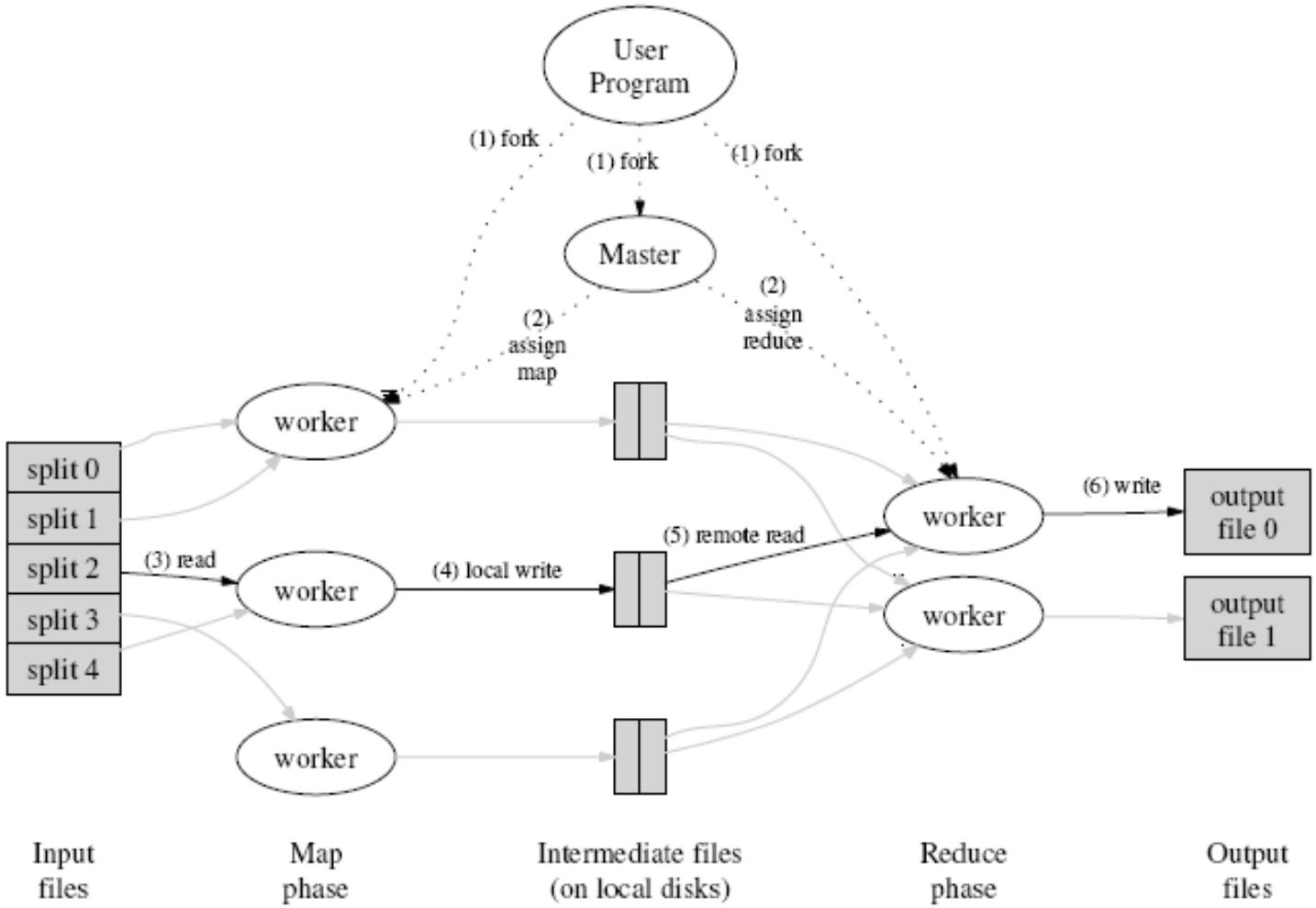
```
map(String input_key, String input_value):
  // input_key: document name
  // input_value: document contents
  for each word w in input_value:
    EmitIntermediate(w, "1");

reduce(String output_key, Iterator intermediate_values):
  // output_key: a word
  // output_values: a list of counts
  int result = 0;
  for each v in intermediate_values:
    result += ParseInt(v);
  Emit(AsString(result));
```

その他の応用例

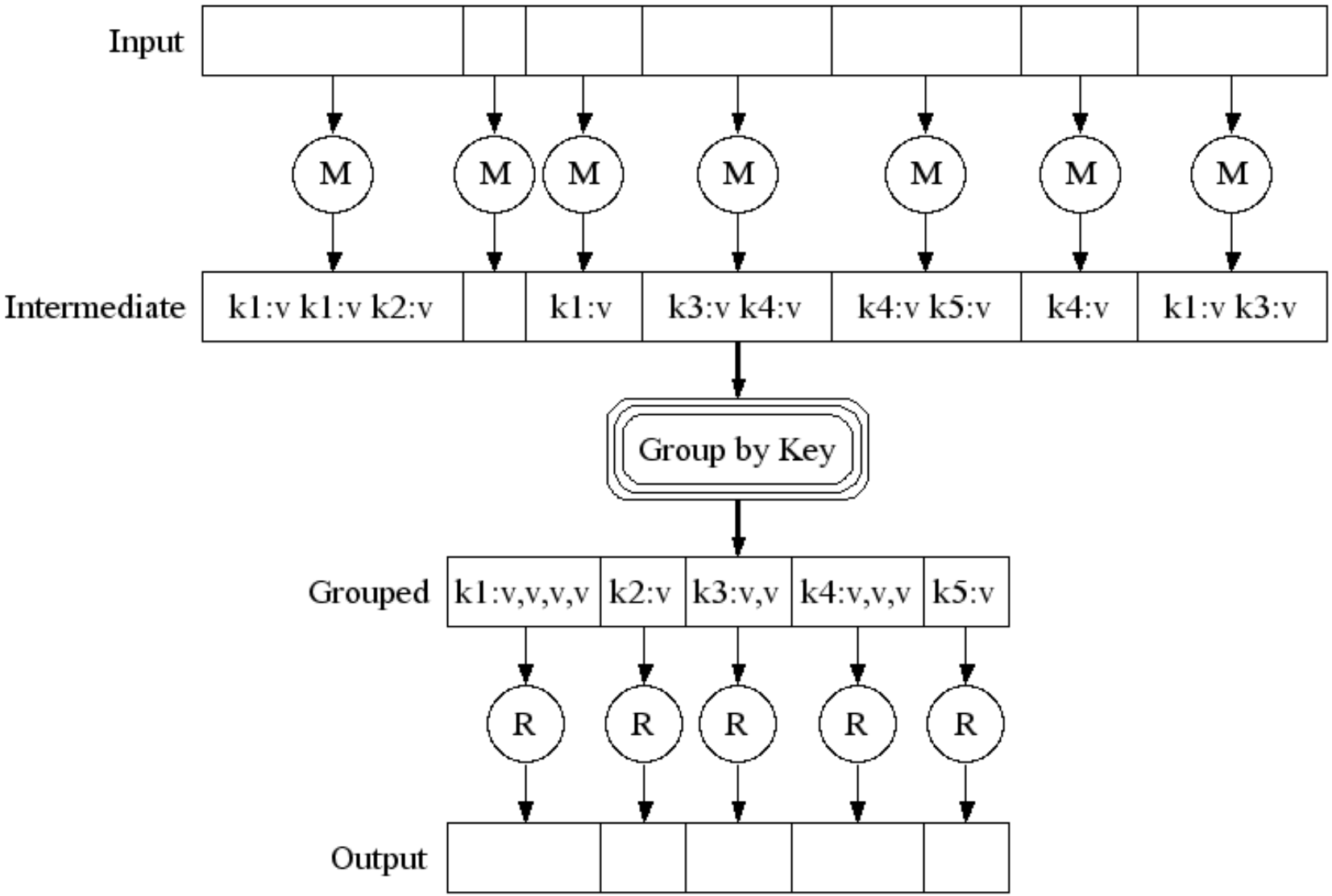
- 分散 grep
  - map: 特定パターンにマッチする行を出力
  - reduce: 何もしない
- URL アクセス頻度カウント
  - map: web access log から $< URL, 1 >$ を出力
  - reduce: 同一 URL の回数を加算し $< URL, count >$ を生成
- reverse web-link graph
  - map: source に含まれる link から、$< target, source >$ を出力
  - reduce: target を link する source list $< target, list(source) >$ を生成
- 逆インデックス
  - map: ドキュメントに含まれる単語から $< word, docID >$ を出力
  - reduce: 特定の単語を含むドキュメントリスト $< word, list(docID) >$ を生成

MapReduce Execution Overview



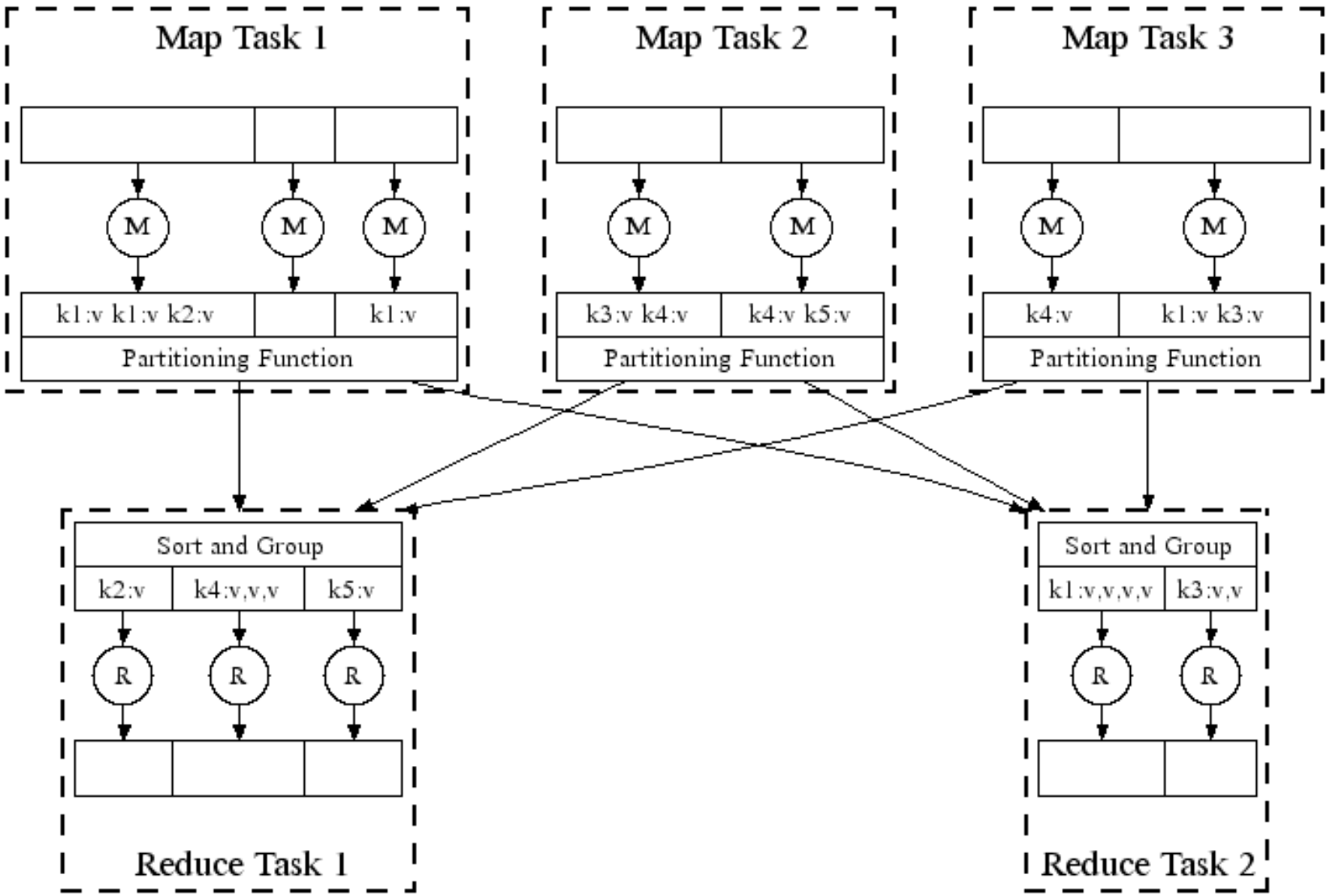
source: MapReduce: Simplified Data Processing on Large Clusters

MapReduce Execution



source: MapReduce: Simplified Data Processing on Large Clusters

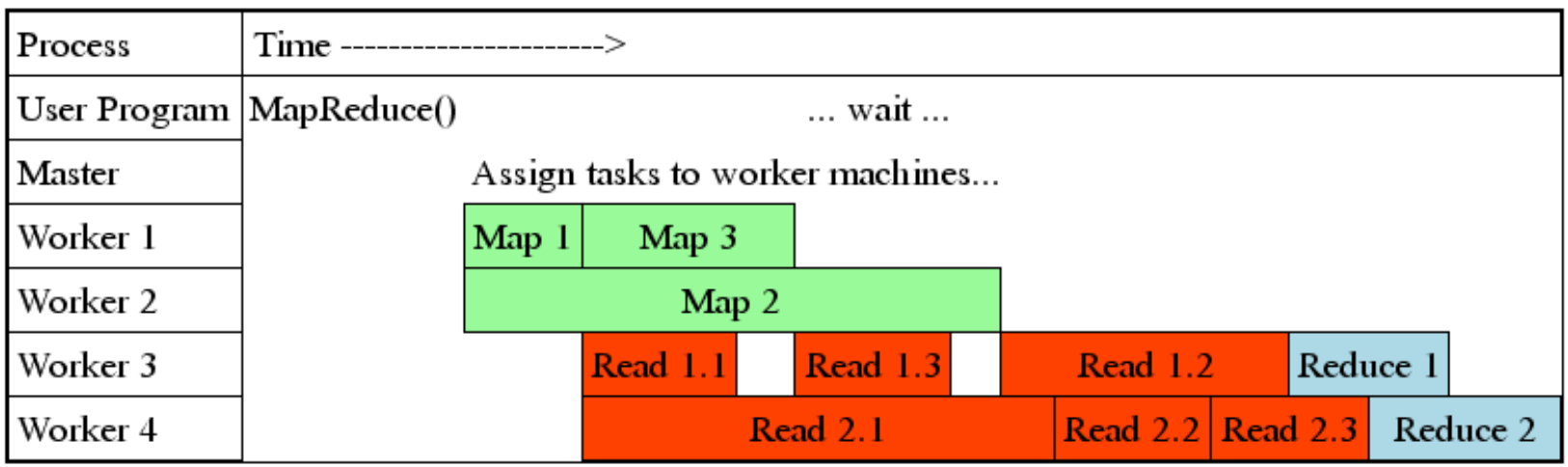
MapReduce Parallel Execution



source: MapReduce: Simplified Data Processing on Large Clusters

Task Granularity and Pipelining

- タスクは細粒度: Map タスク数 >> マシン数
  - 故障復帰時間の低減
  - map 実行をシャフルしてパイプライン実行
  - 実行時に動的ロードバランシング可能
- 典型例: 2,000 台のマシンで、200,000 map/5,000 reduce tasks



source: MapReduce: Simplified Data Processing on Large Clusters

耐故障性

worker の故障

- 定期的なハートビートで故障を検出
- 故障したマシンの map task を再割当てして実行
  - 結果はローカルディスクにあるので終了した task も再実行する
- 実行中の reduce task を再実行
- master が task 終了を監視確認

1800 台中 1600 台のマシンが故障しても正常終了した実績

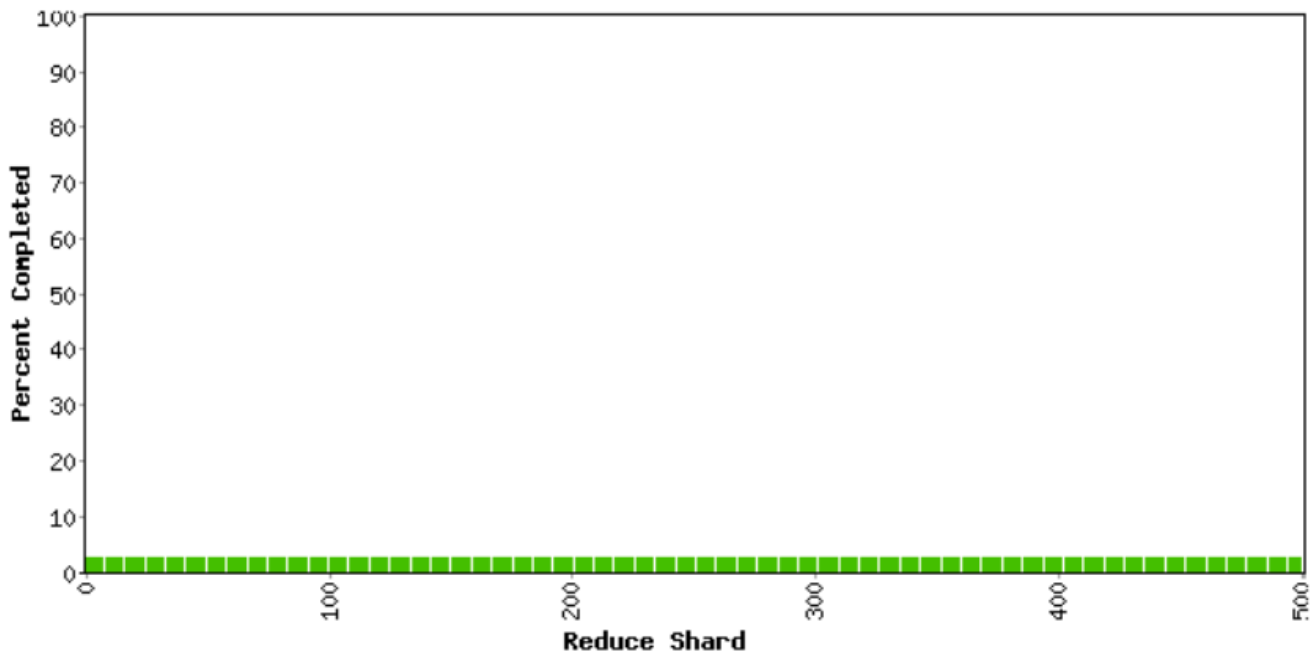
MapReduce status

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 00 min 18 sec

323 workers; 0 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	0	323	878934.6	1314.4	717.0
Shuffle	500	0	323	717.0	0.0	0.0
Reduce	500	0	0	0.0	0.0	0.0



Counters

Variable	Minute
Mapped (MB/s)	72.5
Shuffle (MB/s)	0.0
Output (MB/s)	0.0
doc-index-hits	145825686
docs-indexed	506631
dups-in-index-merge	0
mr-operator-calls	508192
mr-operator-	506631

source: MapReduce: Simplified Data Processing on Large Clusters

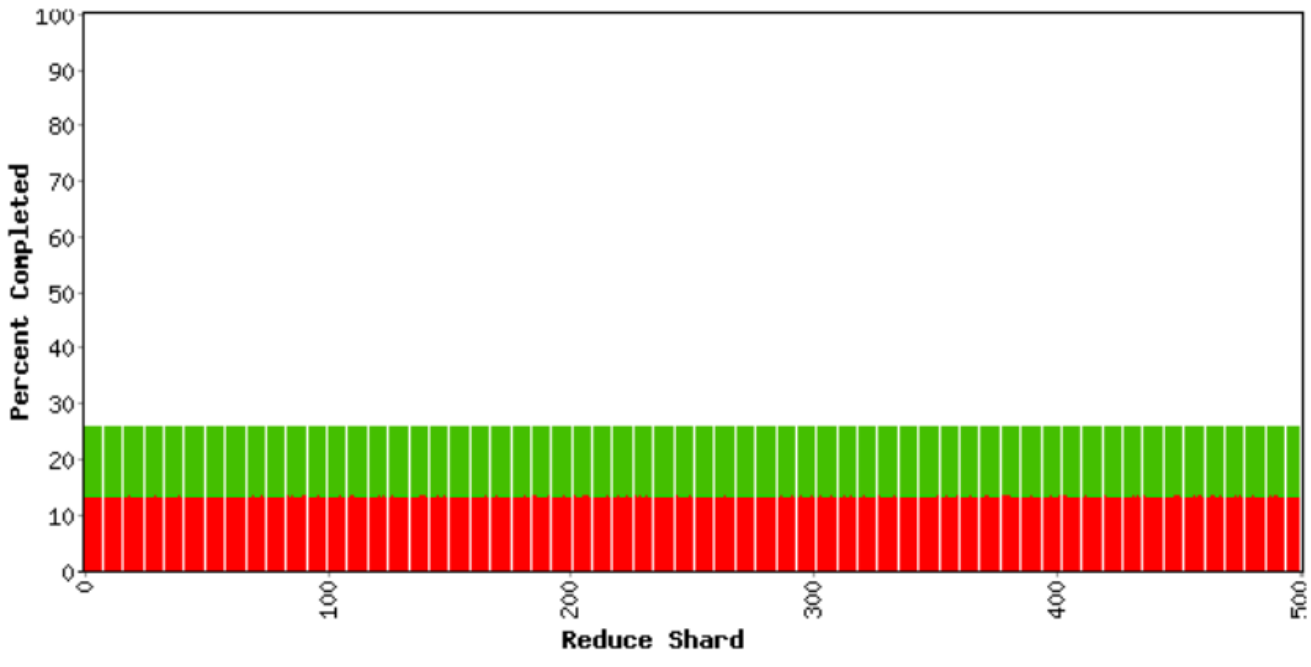
MapReduce status

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 05 min 07 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	1857	1707	878934.6	191995.8	113936.6
Shuffle	500	0	500	113936.6	57113.7	57113.7
Reduce	500	0	0	57113.7	0.0	0.0



Counters

Variable	Minute
Mapped (MB/s)	699.1
Shuffle (MB/s)	349.5
Output (MB/s)	0.0
doc-index-hits	5004411944
docs-indexed	17290135
dups-in-index-merge	0
mr-operator-calls	17331371
mr-operator-outputs	17290135

source: MapReduce: Simplified Data Processing on Large Clusters

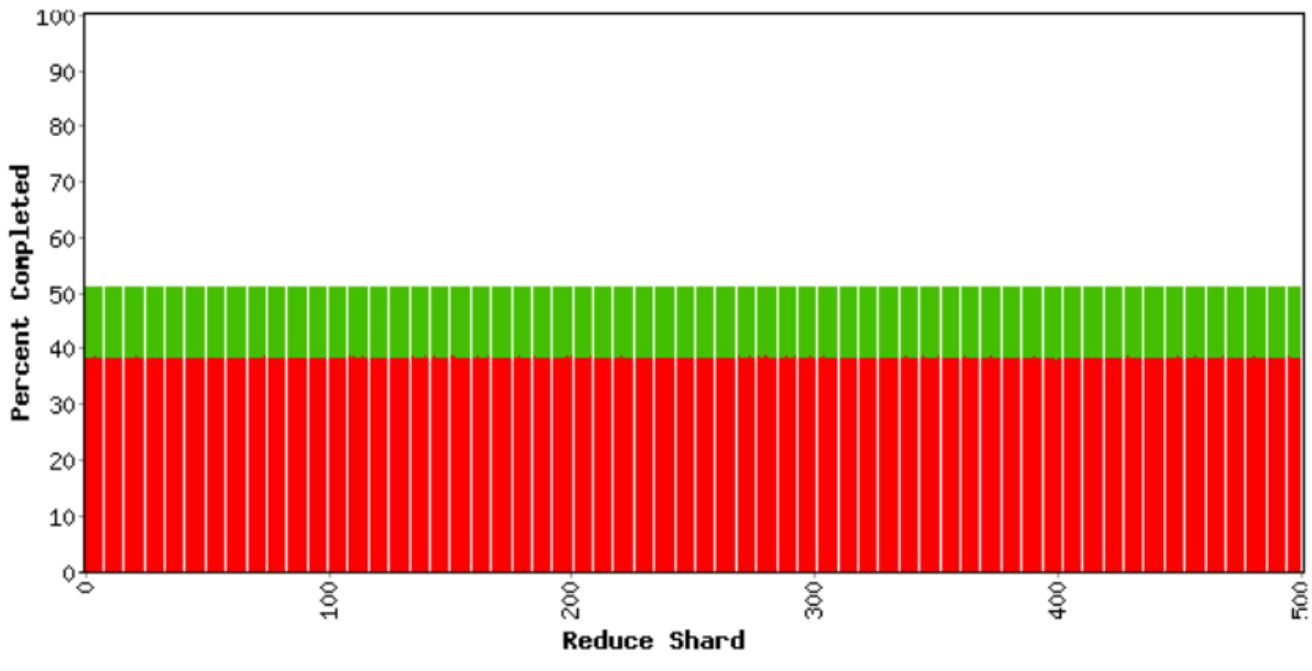
MapReduce status

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 10 min 18 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	5354	1707	878934.6	406020.1	241058.2
Shuffle	500	0	500	241058.2	196362.5	196362.5
Reduce	500	0	0	196362.5	0.0	0.0



Counters

Variable	Minute
Mapped (MB/s)	704.4
Shuffle (MB/s)	371.9
Output (MB/s)	0.0
doc-index-hits	5000364228
docs-indexed	17300709
dups-in-index-merge	0
mr-operator-calls	17342493
mr-operator-outputs	17300709

source: MapReduce: Simplified Data Processing on Large Clusters

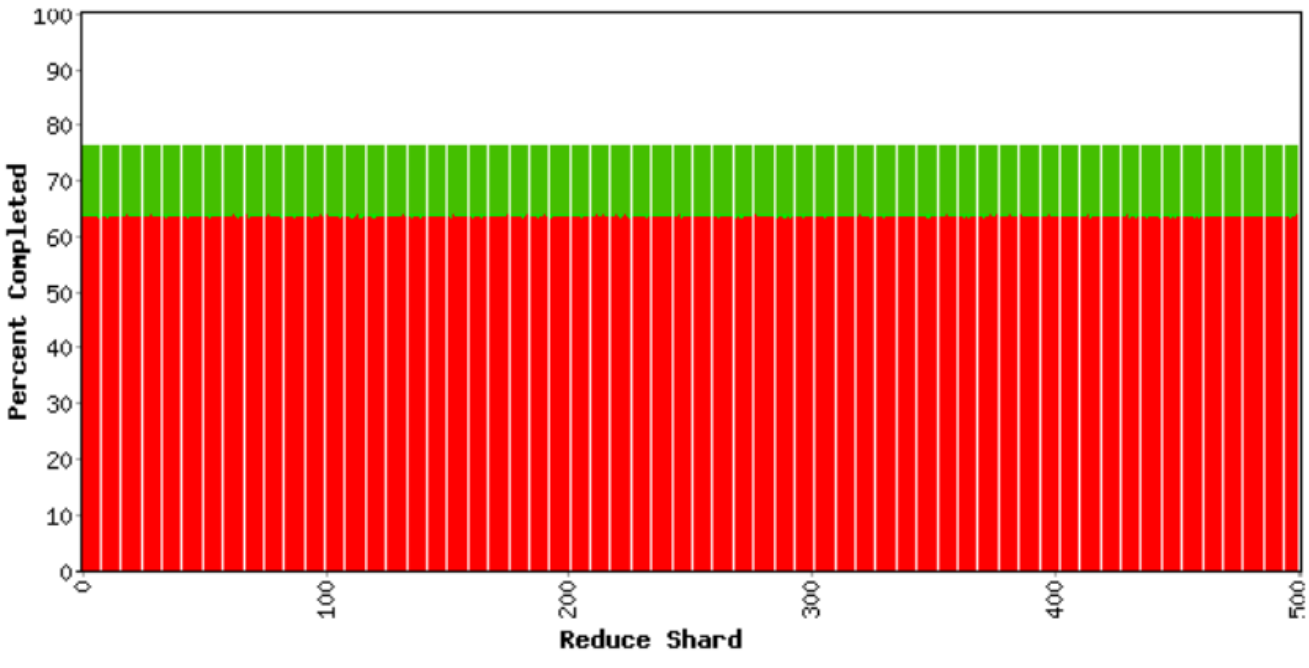
MapReduce status

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 15 min 31 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	8841	1707	878934.6	621608.5	369459.8
Shuffle	500	0	500	369459.8	326986.8	326986.8
Reduce	500	0	0	326986.8	0.0	0.0



Counters

Variable	Minute
Mapped (MB/s)	706.5
Shuffle (MB/s)	419.2
Output (MB/s)	0.0
doc-index-hits	4982870667
docs-indexed	17229926
dups-in-index-merge	0
mr-operator-calls	17272056
mr-operator-outputs	17229926

source: MapReduce: Simplified Data Processing on Large Clusters

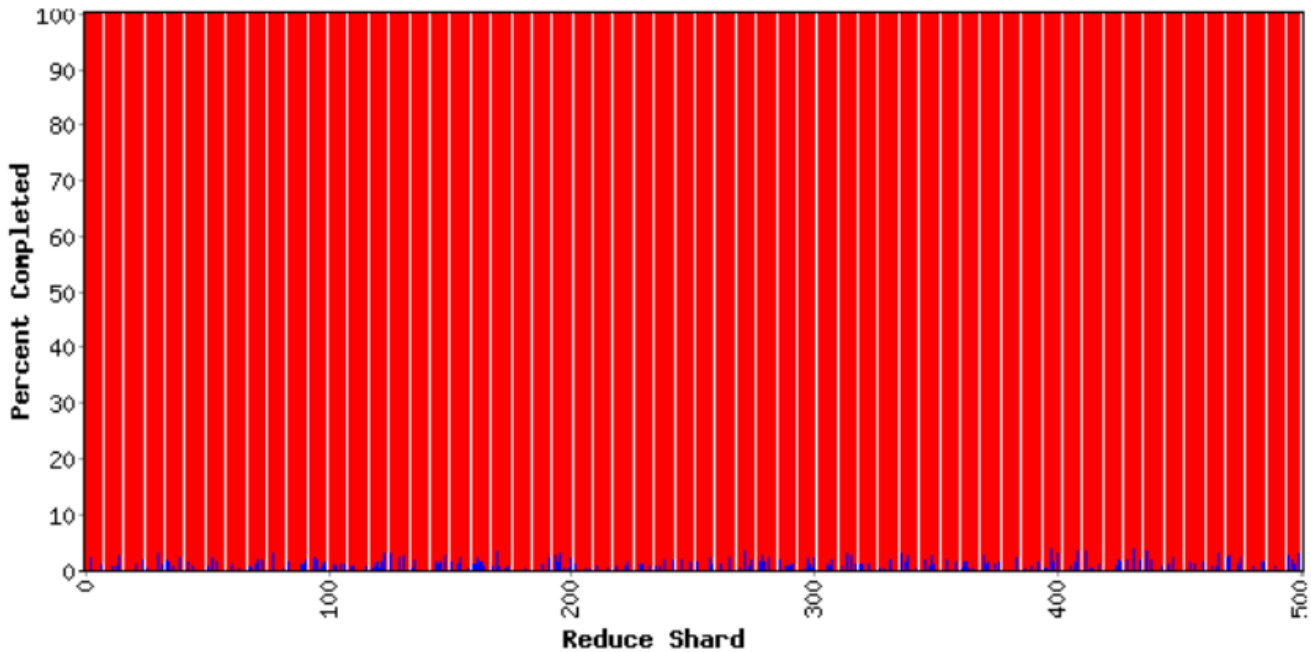
MapReduce status

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 29 min 45 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	195	305	523499.2	523389.6	523389.6
Reduce	500	0	195	523389.6	2685.2	2742.6



Counters

Variable	Minute	
Mapped (MB/s)	0.3	
Shuffle (MB/s)	0.5	
Output (MB/s)	45.7	
doc-index-hits	2313178	10
docs-indexed	7936	
dups-in-index-merge	0	
mr-merge-calls	1954105	
mr-merge-outputs	1954105	

source: MapReduce: Simplified Data Processing on Large Clusters

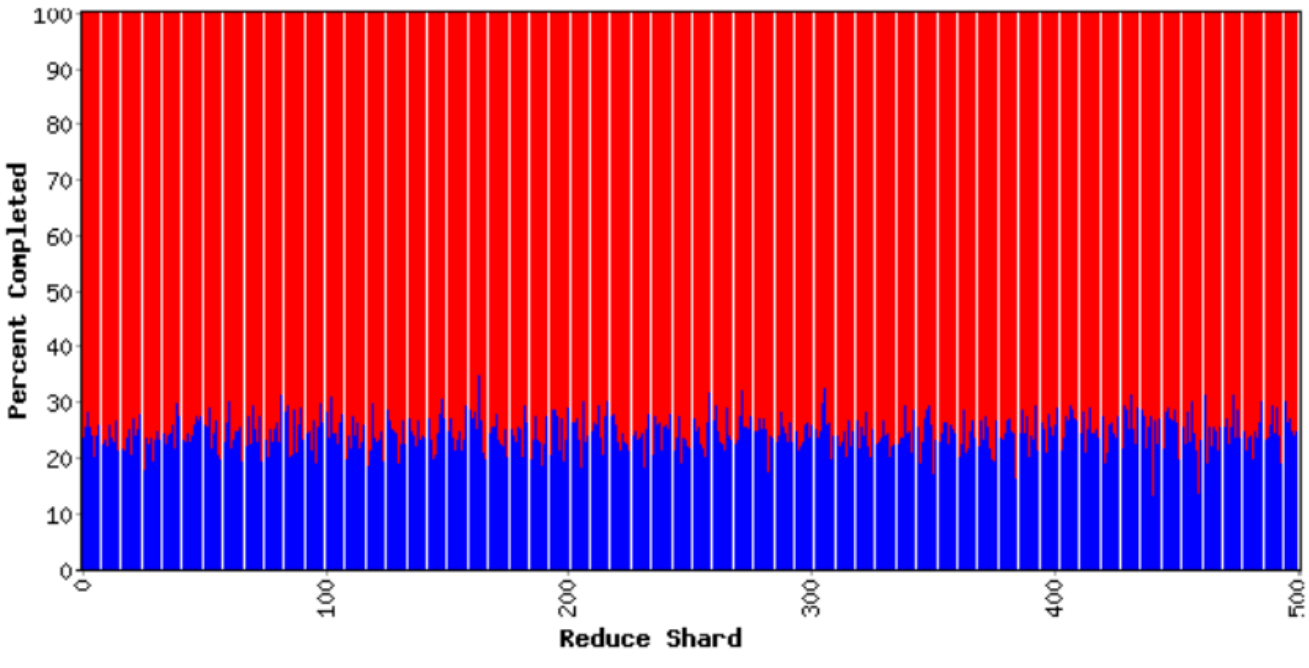
MapReduce status

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 31 min 34 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	133837.8	136929.6



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.1	
Output (MB/s)	1238.8	
doc-index-hits	0	1(
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	51738599	
mr-merge-outputs	51738599	

source: MapReduce: Simplified Data Processing on Large Clusters

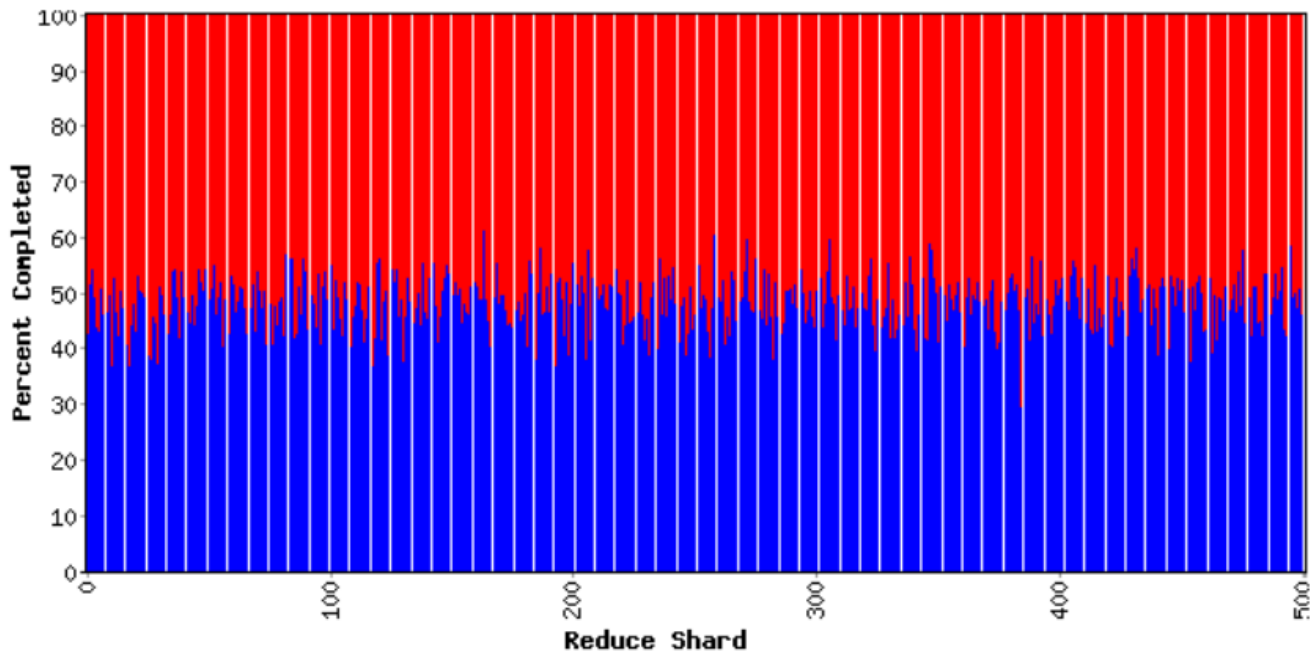
MapReduce status

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 33 min 22 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	263283.3	269351.2



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1225.1	
doc-index-hits	0	1(
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	51842100	
mr-merge-outputs	51842100	

source: MapReduce: Simplified Data Processing on Large Clusters

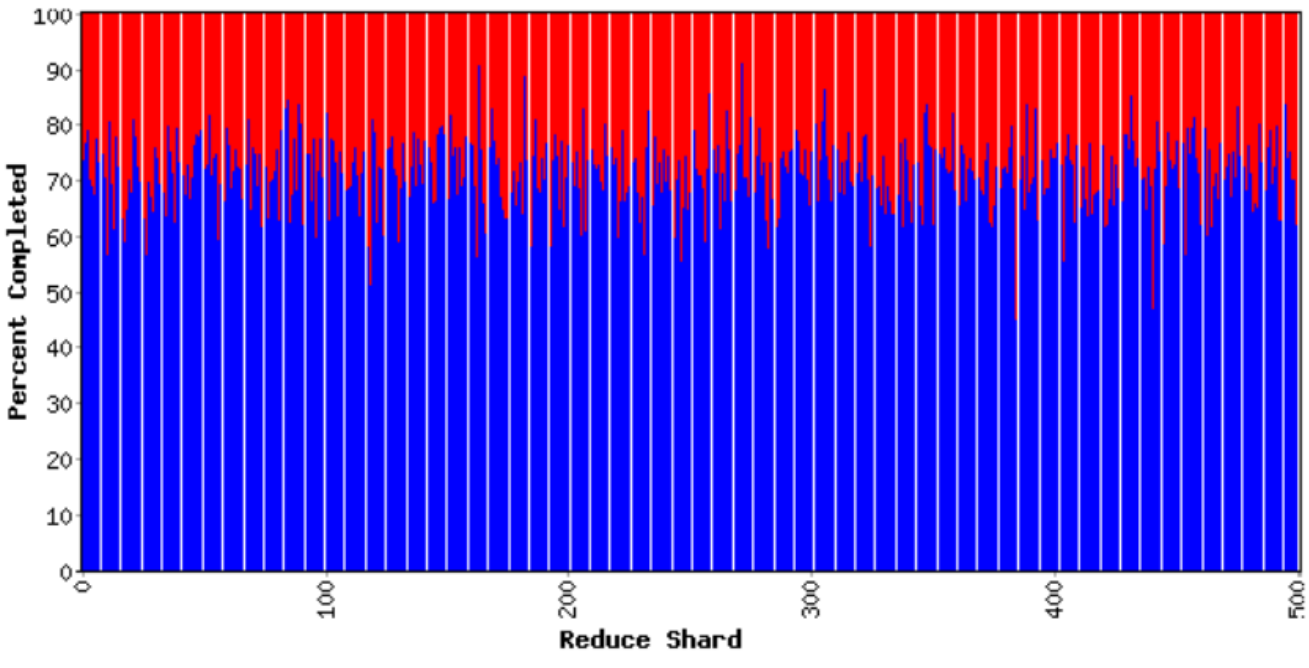
MapReduce status

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 35 min 08 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	390447.6	399457.2



Counters

Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	1222.0
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	51640600
mr-merge-outputs	51640600

source: MapReduce: Simplified Data Processing on Large Clusters

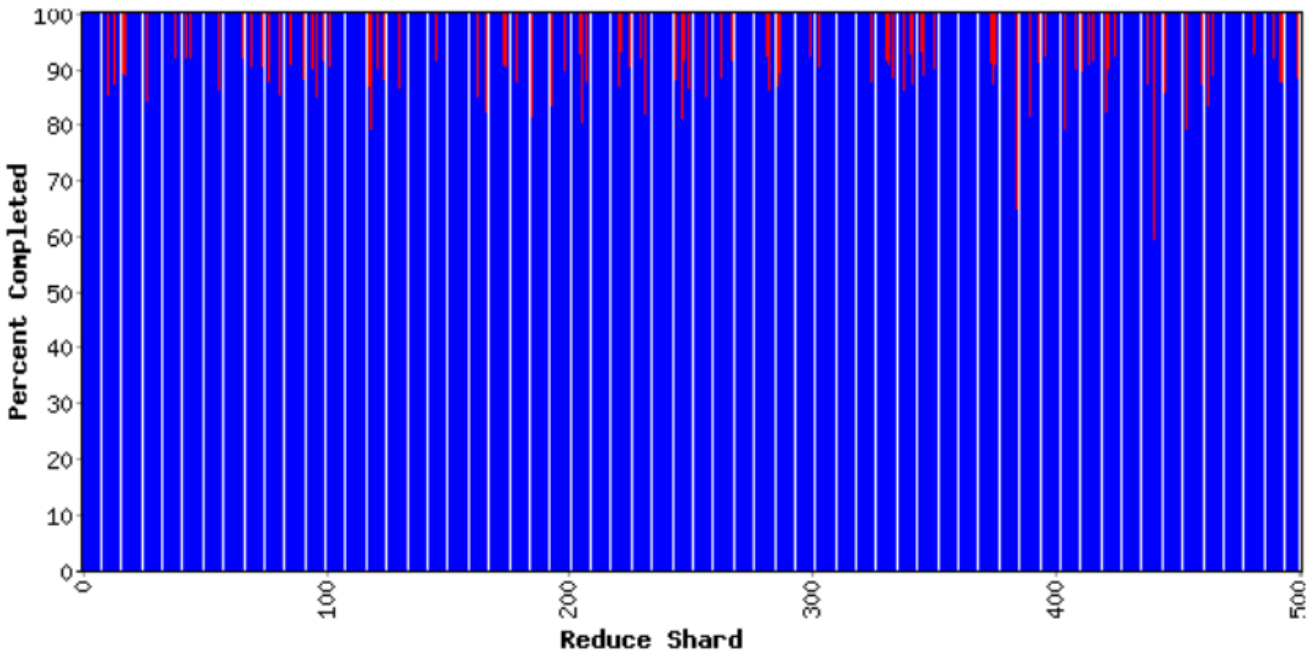
MapReduce status

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 37 min 01 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	520468.6	520468.6
Reduce	500	406	94	520468.6	512265.2	514373.3



Counters

Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	849.5
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	35083350
mr-merge-outputs	35083350

source: MapReduce: Simplified Data Processing on Large Clusters

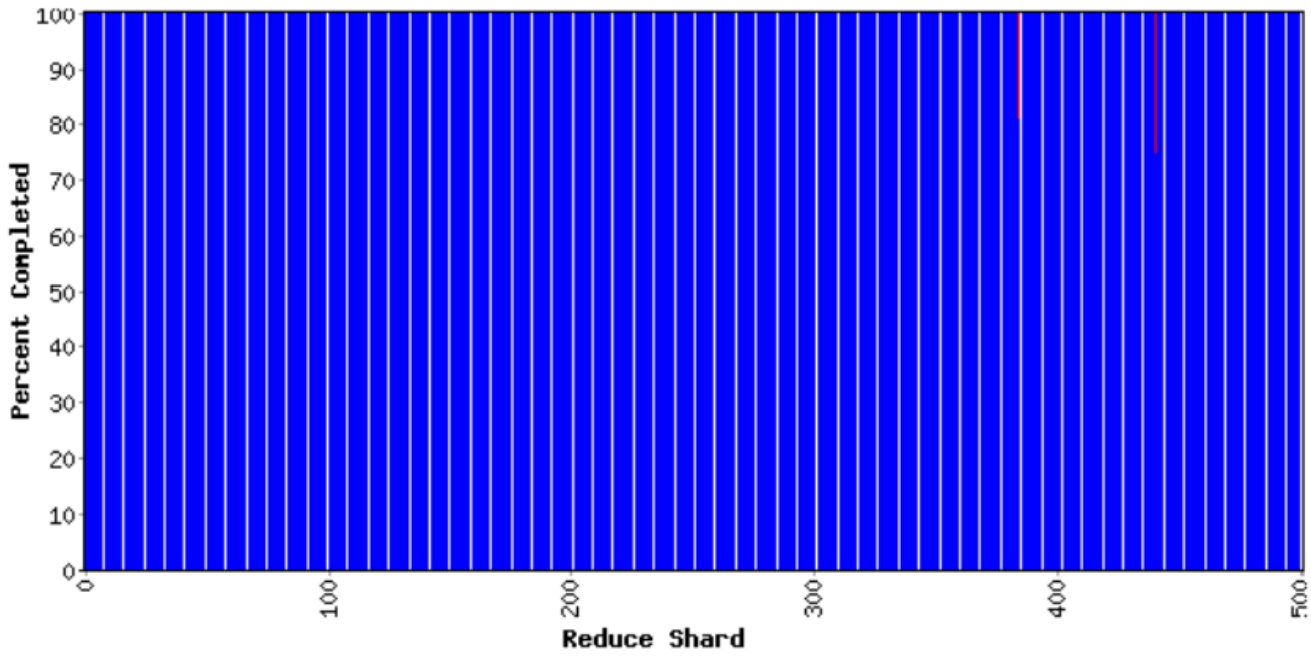
MapReduce status

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 38 min 56 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	519781.8	519781.8
Reduce	500	498	2	519781.8	519394.7	519440.7



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	9.4	
doc-index-hits	0	1056
docs-indexed	0	1
dups-in-index-merge	0	
mr-merge-calls	394792	1
mr-merge-outputs	394792	1

source: MapReduce: Simplified Data Processing on Large Clusters

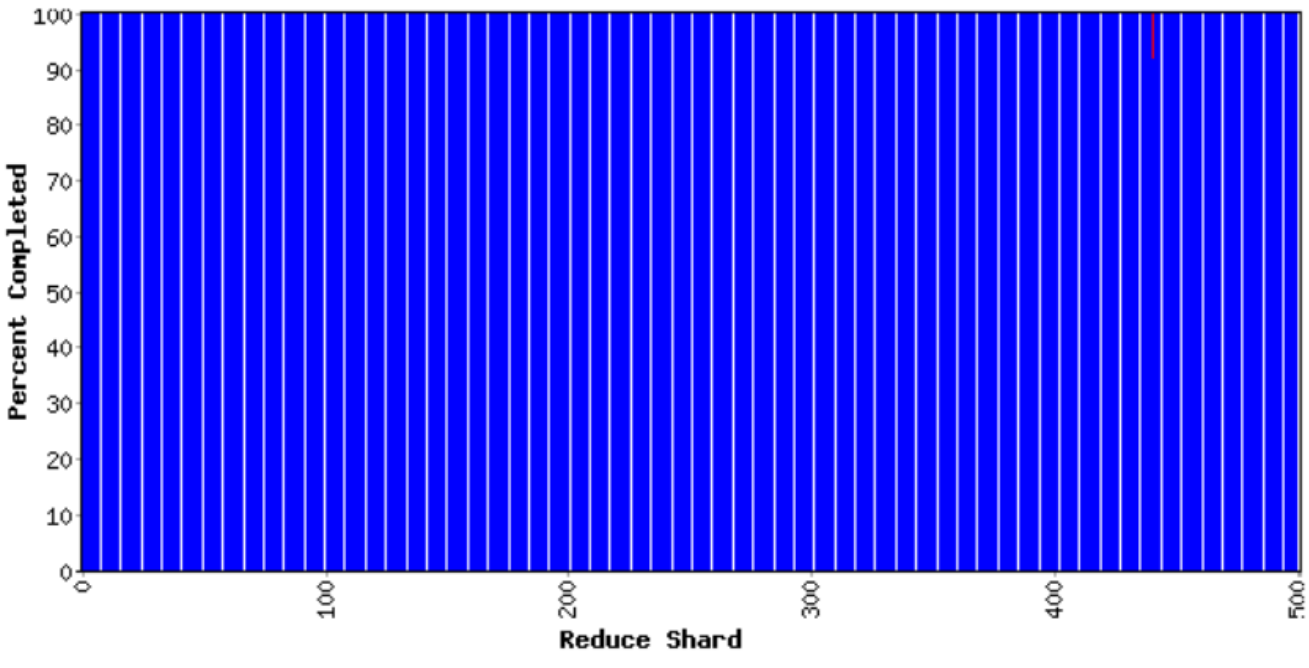
MapReduce status

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 40 min 43 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	519774.3	519774.3
Reduce	500	499	1	519774.3	519735.2	519764.0



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1.9	
doc-index-hits	0	105
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	73442	
mr-merge-outputs	73442	

source: MapReduce: Simplified Data Processing on Large Clusters

冗長実行

遅い worker がいると終了時間に大きな影響

- 別ジョブの影響
- ディスクのソフトエラー
- その他の要因: CPU cache が disable されていたケースも!

解決策: 全体処理終了近くで、backup tasks を起動

- 早く終了した task の結果を採用

ジョブ終了時間の大幅短縮に成功

ローカリティの最適化

Master のスケジューリングポリシー

- GFS に入力ファイルブロックの複製の位置を問い合わせ
- 入力を 64MB 単位 (GFS block size) に分割
- 入力データの複製があるマシンまたはラックに map task を割当てる

効果: 何千台のマシンがローカルなディスクから入力を読み込み

- さもなければ、ラックのスイッチがボトルネックになる

不良レコードのスキップ

Map/Reduce 機能が時に特定のレコードの処理でクラッシュすることがある

- 原因はバグ: デバッグして問題解決するのが最善だが、そう出来ない場合も多い
- Segmentation Fault の発生時
  - シグナルハンドラからマスターに UDP パケットを送信
  - 処理中のレコード番号を通知
- Master は同じレコードの不良が 2 回起こると
  - 次の worker にそのレコードをスキップするよう指示

効果: サードパーティ製ライブラリのバグ回避

その他の最適化

- 各 reduce パーティション内でソートされた順序を保証
- 中間データの圧縮
- Combiner: 冗長な結果を集約してネットワーク使用量削減
- デバッグやテスト用のローカル実行環境
- ユーザ定義のカウンタが利用可能

性能評価

1800 台のマシンクラスタを使った性能評価を実施

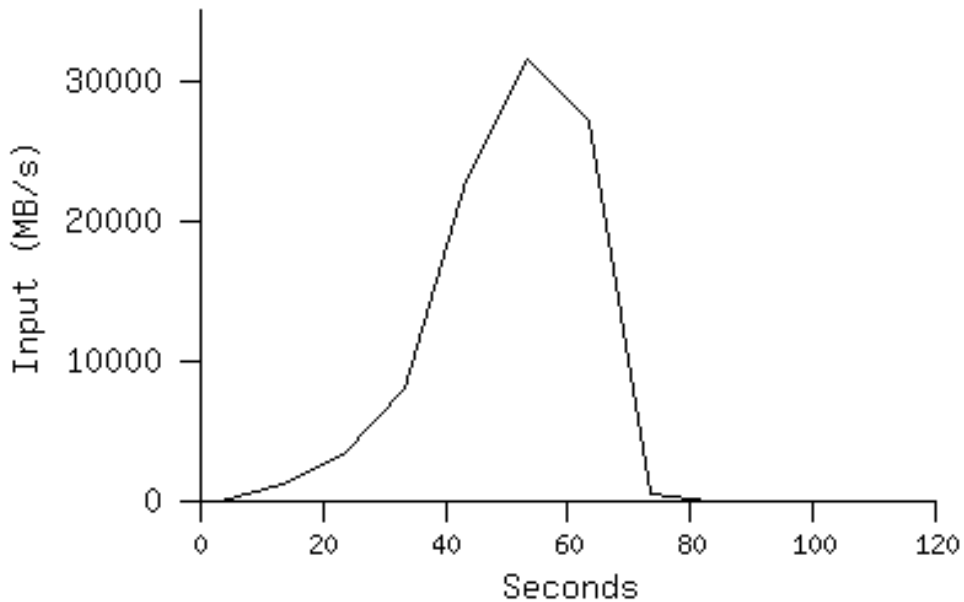
- 4GB of memory
- Dual-processor 2GHz Xeons with Hyperthreading
- Dual 160GB IDE disks
- Gigabit Ethernet per machine
- Bisection bandwidth approximately 100Gbps

2 種類のベンチマーク

- MR_Grep: 10^{10} 100B レコードをスキャンして特定のパターンを抜き出す
- MR_Sort: 10^{10} 100B レコードをソート (TeraSort ベンチマークのモデルを利用)

MR_Grep

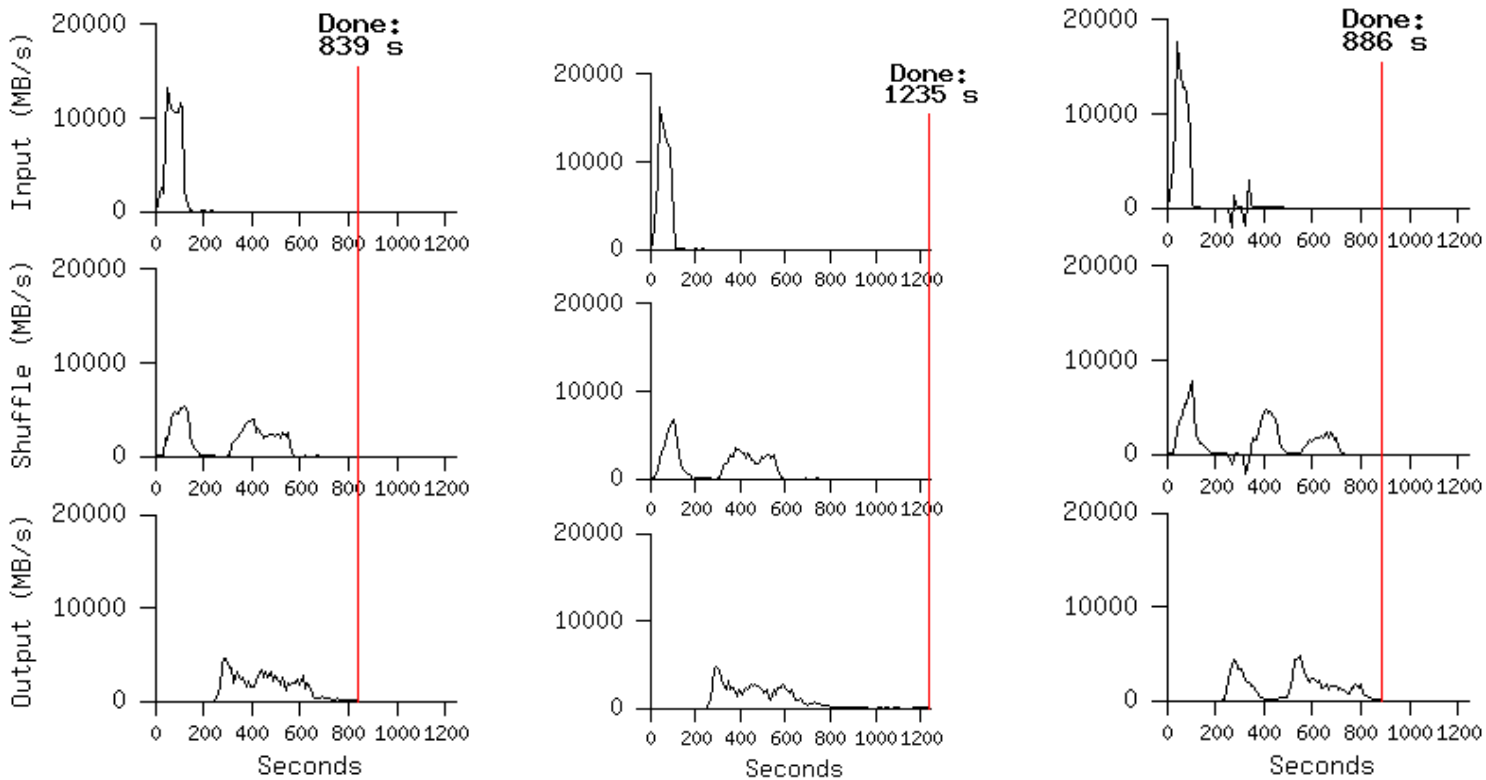
- ローカリティ最適化効果
  - 1800 台のマシンが 1TB のデータを最大 31GB/s で読み込み
  - さもなければ、ラックスイッチがボトルネックで 10GB/s 程度
- 短いジョブでは始動時のオーバヘッドが大きい



source: MapReduce: Simplified Data Processing on Large Clusters

MR_Sort

- backup task による完了時間の大幅短縮
- 耐故障性の実現



Normal(left) No backup tasks(middle) 200 processes killed(right)

source: MapReduce: Simplified Data Processing on Large Clusters

Hadoop MapReduce

- Hadoop
  - Apache のオープンソースソフトウェアプロジェクト
  - Java ソフトウェアフレームワーク
  - Google の GFS 分散ファイルシステムや Mapreduce を実装
  - 大規模データ解析プラットフォームとして広く利用されている
- Hadoop MapReduce
  - Java による実装
  - MapReduce 処理のためのサーバ、ライブラリ
  - Master/Slave アーキテクチャ

WordCount in Hadoop MapReduce (1/3)

```
package org.myorg;

import java.io.IOException;
import java.util.*;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.conf.*;
import org.apache.hadoop.io.*;
import org.apache.hadoop.mapred.*;
import org.apache.hadoop.util.*;

public class WordCount {

   public static class Map extends MapReduceBase implements Mapper<LongWritable,
         Text, Text, IntWritable> {
     private final static IntWritable one = new IntWritable(1);
     private Text word = new Text();

     public void map(LongWritable key, Text value, OutputCollector<Text, IntWritable> output,
           Reporter reporter) throws IOException {
       String line = value.toString();
       StringTokenizer tokenizer = new StringTokenizer(line);
       while (tokenizer.hasMoreTokens()) {
         word.set(tokenizer.nextToken());
         output.collect(word, one);
       }
     }
   }
```

WordCount in Hadoop MapReduce (2/3)

```
public static class Reduce extends MapReduceBase implements Reducer<Text, IntWritable,
      Text, IntWritable> {
  public void reduce(Text key, Iterator<IntWritable> values, OutputCollector<Text, IntWritable>
        output, Reporter reporter) throws IOException {
    int sum = 0;
    while (values.hasNext()) {
      sum += values.next().get();
    }
    output.collect(key, new IntWritable(sum));
  }
}
```

WordCount in Hadoop MapReduce (3/3)

```
  public static void main(String[] args) throws Exception {
    JobConf conf = new JobConf(WordCount.class);
    conf.setJobName("wordcount");

    conf.setOutputKeyClass(Text.class);
    conf.setOutputValueClass(IntWritable.class);

    conf.setMapperClass(Map.class);
    conf.setCombinerClass(Reduce.class);
    conf.setReducerClass(Reduce.class);

    conf.setInputFormat(TextInputFormat.class);
    conf.setOutputFormat(TextOutputFormat.class);

    FileInputFormat.setInputPaths(conf, new Path(args[0]));
    FileOutputFormat.setOutputPath(conf, new Path(args[1]));

    JobClient.runJob(conf);
  }
}
```

WordCount in Ruby

Ruby で MapReduce ぽい処理をしてみる

```
% cat wc-data.txt
Hello World Bye World
Hello Hadoop Goodbye Hadoop
% cat wc-data.txt | ruby wc-map.rb | sort | ruby wc-reduce.rb
bye     1
goodbye 1
hadoop  2
hello   2
world   2
```

WordCount in Ruby: Map

```
#!/usr/bin/env ruby
#
# word-count map task: input <text>, output a list of <word, 1>

ARGF.each_line do |line|
  words = line.split(/\W+/)
  words.each do |word|
    if word.length < 20 && word.length > 2
      printf "%s\t1\n", word.downcase
    end
  end
end
```

WordCount in Ruby: Reduce

```
#!/usr/bin/env ruby
#
# word-count reduce task: input a list of <word, count>, output <word, count>
# assuming the input is sorted by key
current_word = nil
current_count = 0
word = nil

ARGF.each_line do |line|
  word, count = line.split

  if current_word == word
    current_count += count.to_i
  else
    if current_word != nil
      printf "%s\t%d\n", current_word, current_count
    end
    current_word = word
    current_count = count.to_i
  end
end
if current_word == word
  printf "%s\t%d\n", current_word, current_count
end
```

MapReduce まとめ

- MapReduce: 並列分散処理の抽象化モデル
- 大規模データ処理を大幅に簡略化
- 使い易い、楽しい
  - 並列処理部分の詳細をシステムにまかせ問題解決に専念できる
- Google 内部で検索インデックス作成をはじめさまざまな応用

補足

- Google の MapReduce 実装は非公開
- Hadoop: Apache プロジェクトのオープンソース MapReduce 実装

最終レポートについて

- A, B からひとつ選択
  - A. ウィキペディア日本語版の Pageview ランキング
  - B. 自由課題
- 8 ページ以内
- pdf ファイルで提出
- 提出〆切: 2014 年 7 月 28 日 (月) 23:59

最終レポート 選択テーマ

A. ウィキペディア日本語版の Pageview ランキング

- ねらい: 実データから人気キーワードを抽出し時間変化を観測
- データ: ウィキペディア日本語版の Pageview データ
- 提出項目
  - A-1 Pageview カウント分布調査
    - 各ページの 1 週間分のリクエスト総数を集計し、分布を CCDF でプロット
  - A-2 各日および 1 週間合計からリクエスト数トップ 10 を抽出
    - トップ 10 の結果を表にする
  - A-3 週間トップ 10 についてランキングの推移をプロット
    - ランキング変化が分かり易いよう時間粒度を考え図を工夫する
  - A-4 オプション解析: その他の自由解析
  - A-5 考察: データから読みとれることを考察

B. 自由課題

- 授業内容と関連するテーマを自分で選んでレポート
- 必ずしもネットワーク計測でなくてもよいが、何らかのデータ解析を行い、考察すること

最終レポートは考察を重視する

課題 A. ウィキペディア日本語版の Pageview ランキング

データ: ウィキペディア日本語版のデータ Pageview データ

- wikimedia が提供するデータからウィキペディア日本語版だけを抜き出したもの。
- 元データ情報: http://dumps.wikimedia.org/other/pagecounts-raw/
- 課題用 Pageview データ: 20140616-22.zip (609MB 解凍後 3GB)
  - 1 時間毎の Pageview データ 1 週間分 (2014 年 6 月 16 日-22 日)
- オプションデータセット: 20140601-15.zip (1.3GB 解凍後 6.3GB)
  - オプション解析で利用可能な追加データ (2014 年 6 月 1 日-15 日)

データフォーマット

- project encoded_pagetitle requests size
  - project: wikimedia のプロジェクト名 (課題用データでは全て"ja")
  - encoded_pagetitle: URI エンコードされたページタイトル
  - requests: ページのリクエスト回数
  - size: ページのバイト数

```
% head -n 10 20140616-22/pagecounts-20140616-00
ja $ 1 0
ja $10 1 8922
ja %22B%22ORDERLESS 1 13777
ja %22BLUE%22_A_TRIBUTE_TO_YUTAKA_OZAKI 1 21159
ja %22HAPPY%22_Coming_Century,_20th_Century_Forever 1 21326
ja %22LUCKY%22_20th_Century,_Coming_Century_to_be_continued... 1 0
ja %22X%22_plosion_GUNDAM_SEED 2 50386
ja %26C 1 16485
ja %26_(%E4%B8%80%E9%9D%92%E7%AA%88%E3%81%AE%E3%82%A2%E3%83%AB%E3%83%90%E3%83%A0) 1 12635
ja %26_(%E6%BC%AB%E7%94%BB) 1 0

% head -n 10 20140616-22/pagecounts-20140616-00 | ./urldecode.rb
ja "$" 1 0
ja "$10" 1 8922
ja ""B"ORDERLESS" 1 13777
ja ""BLUE"_A_TRIBUTE_TO_YUTAKA_OZAKI" 1 21159
ja ""HAPPY"_Coming_Century,_20th_Century_Forever" 1 21326
ja ""LUCKY"_20th_Century,_Coming_Century_to_be_continued..." 1 0
ja ""X"_plosion_GUNDAM_SEED" 2 50386
ja "&C" 1 16485
ja "&_(一青窈のアルバム)" 1 12635
ja "&_(漫画)" 1 0
```

タイトルのデコードスクリプト

- タイトルはパーセントエンコードされている
  - ruby の CGI.unescape() で UTF-8 に変換できる

```
#!/usr/bin/env ruby

require 'cgi'

re = /^([\w\.]+)\s+(\S+)\s+(\d+)\s+(\d+)/

ARGF.each_line do |line|
  if re.match(line)
    project, title, requests, bytes = $~.captures
    decoded_title = CGI.unescape(title)
    print "#{project} \"#{decoded_title}\" #{requests} #{bytes}\n"
  end
end
```

課題 A Pageview ランキング補足

- A-1 Pageview カウント分布調査
  - 各ページの 1 週間分のリクエスト総数を集計し、分布を CCDF でプロット
  - X 軸はリクエスト数、Y 軸は CCDF、log-log でプロット
- A-2 各日および 1 週間合計からリクエスト数トップ 10 を抽出
  - トップ 10 の結果を表にする

```
rank  6/16 6/17 6/18 6/19 6/20 6/21 6/22 total
  1   "a"  "b"  "c"  "d"  "e"  "f"  "g"  "x"
  2   "h"  "i"  "j"  "k"  "l"  "m"  "n"  "y"
...
 10   "o"  "p"  "q"  "r"  "s"  "t"  "u"  "z"
```

- A-3 週間トップ 10 についてランキングの推移をプロット
  - X 軸に時間、Y 軸にランキングをとる
  - ランキング変化が分かり易いよう時間粒度やランキングの見せ方を考え図を工夫する

前回の演習: PageRank

```
% cat sample-links.txt
# PageID: OutLinks
1:      2       3       4       5       7
2:      1
3:      1       2
4:      2       3       5
5:      1       3       4       6
6:      1       5
7:      5
% ruby pagerank.rb -f 1.0 sample-links.txt
reading input...
initializing... 7 pages dampingfactor:1.00 thresh:0.000001
iteration:1  diff_sum:0.661905 rank_sum: 1.000000
iteration:2  diff_sum:0.383333 rank_sum: 1.000000
...
iteration:20  diff_sum:0.000002 rank_sum: 1.000000
iteration:21  diff_sum:0.000001 rank_sum: 1.000000
[1] 1 0.303514
[2] 5 0.178914
[3] 2 0.166134
[4] 3 0.140575
[5] 4 0.105431
[6] 7 0.060703
[7] 6 0.044728
```

前回の演習: PageRank code (1/4)

```
require 'optparse'

d = 0.85   # damping factor (recommended value: 0.85)
thresh = 0.000001 # convergence threshold

OptionParser.new {|opt|
  opt.on('-f VAL', Float) {|v| d = v}
  opt.on('-t VAL', Float) {|v| thresh = v}
  opt.parse!(ARGV)
}

outdegree = Hash.new # outdegree[id]: outdegree of each page
inlinks = Hash.new # inlinks[id][src0, src1, ...]: inlinks of each page
rank = Hash.new # rank[id]: pagerank of each page
last_rank = Hash.new # last_rank[id]: pagerank at the last stage
dangling_nodes = Array.new # dangling pages: pages without outgoing link

# read a page-link file: each line is "src_id  dst_id_1 dst_id_2 ..."
ARGF.each_line do |line|
  pages = line.split(/\D+/)  # extract list of numbers
  next if line[0] == ?# || pages.empty?

  src = pages.shift.to_i  # the first column is the src
  outdegree[src] = pages.length
  if outdegree[src] == 0
    dangling_nodes.push src
  end
  pages.each do |pg|
    dst = pg.to_i
    inlinks[dst] ||= []
    inlinks[dst].push src
  end
end
```

前回の演習: PageRank code (2/4)

```
# initialize
# sanity check: if dst node isn't defined as src, create one as a dangling node
inlinks.each_key do |j|
  if !outdegree.has_key?(j)
    # create the corresponding src as a dangling node
    outdegree[j] = 0
    dangling_nodes.push j
  end
end

n = outdegree.length # total number of nodes
# initialize the pagerank of each page with 1/n
outdegree.each_key do |i|  # loop through all pages
  rank[i] = 1.0 / n
end
$stderr.printf " %d pages dampingfactor:%.2f thresh:%f\n", n, d, thresh
```

前回の演習: PageRank code (3/4)

```
# compute pagerank by power method
k = 0 # iteration number
begin
  rank_sum = 0.0  # sum of pagerank of all pages: should be 1.0
  diff_sum = 0.0  # sum of differences from the last round
  last_rank = rank.clone  # copy the entire hash of pagerank

  # compute dangling ranks
  danglingranks = 0.0
  dangling_nodes.each do |i| # loop through dangling pages
    danglingranks += last_rank[i]
  end

  # compute page rank
  outdegree.each_key do |i|  # loop through all pages
    inranks = 0.0
    # for all incoming links for i, compute
    #    inranks = sum (rank[j]/outdegree[j])
    if inlinks[i] != nil
      inlinks[i].each do |j|
        inranks += last_rank[j] / outdegree[j]
      end
    end

    rank[i] = d * (inranks + danglingranks / n) + (1.0 - d) / n
    rank_sum += rank[i]

    diff = last_rank[i] - rank[i]
    diff_sum += diff.abs
  end

  k += 1
  $stderr.printf "iteration:%d  diff_sum:%f rank_sum: %f\n", k, diff_sum, rank_sum
end while diff_sum > thresh
```

前回の演習: PageRank code (4/4)

```
# print pagerank in the decreasing order of the rank
# format: [position]  id  pagerank
i = 0
rank.sort_by{|k, v| -v}.each do |k, v|
  i += 1
  printf "[%d] %d %f\n", i, k, v
end
```

まとめ

第 13 回 スケールする計測と解析

- 大規模計測
- クラウド技術
- MapReduce
- 演習: MapReduce

次回予定

第 14 回 まとめ (7/14)

- これまでのまとめ
- インターネット計測とプライバシー