インターネット計測とデータ解析 第14回

長 健二朗

2012年7月13日

前回のおさらい

データマイニング

- パターン抽出
- クラス分類
- クラスタリング
- 演習: クラスタリング

今日のテーマ

スケールする計測と解析

- 大規模計測
- MapReduce
- 分散並列処理
- クラウド技術
- インターネット計測とプライバシー
- 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)

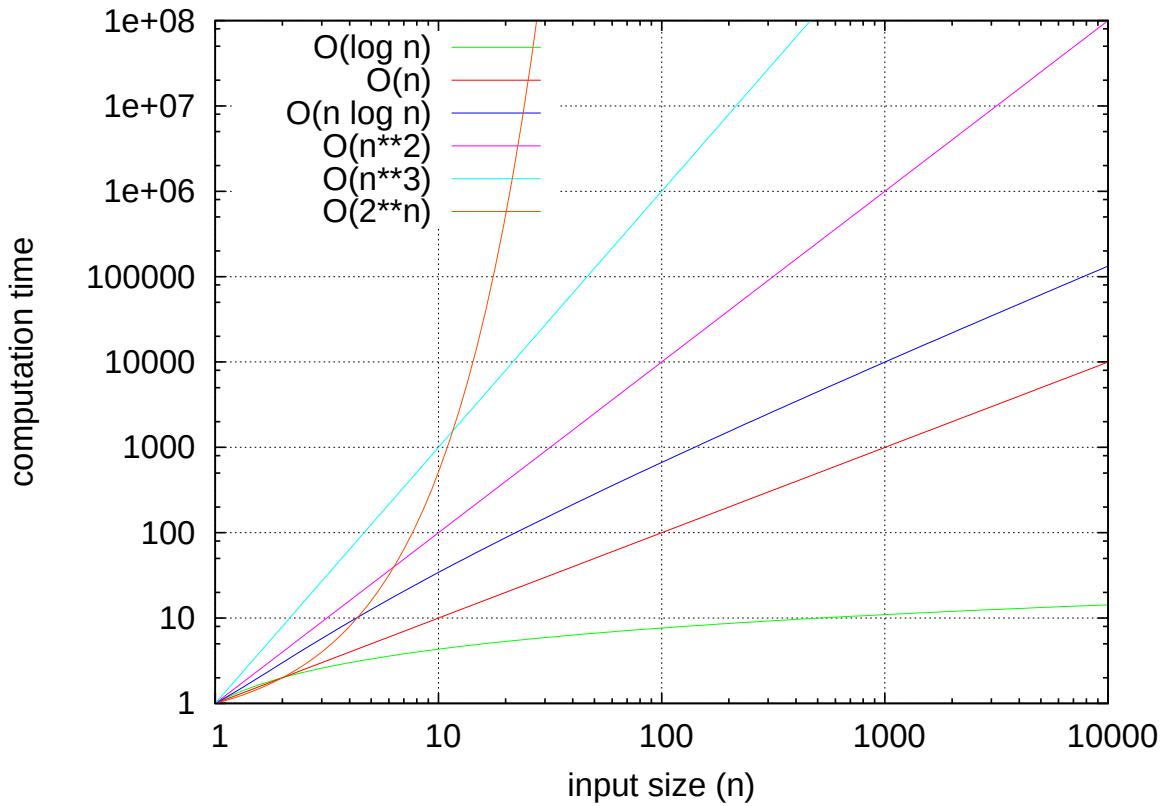
- 平均計算量
- 最悪計算量

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

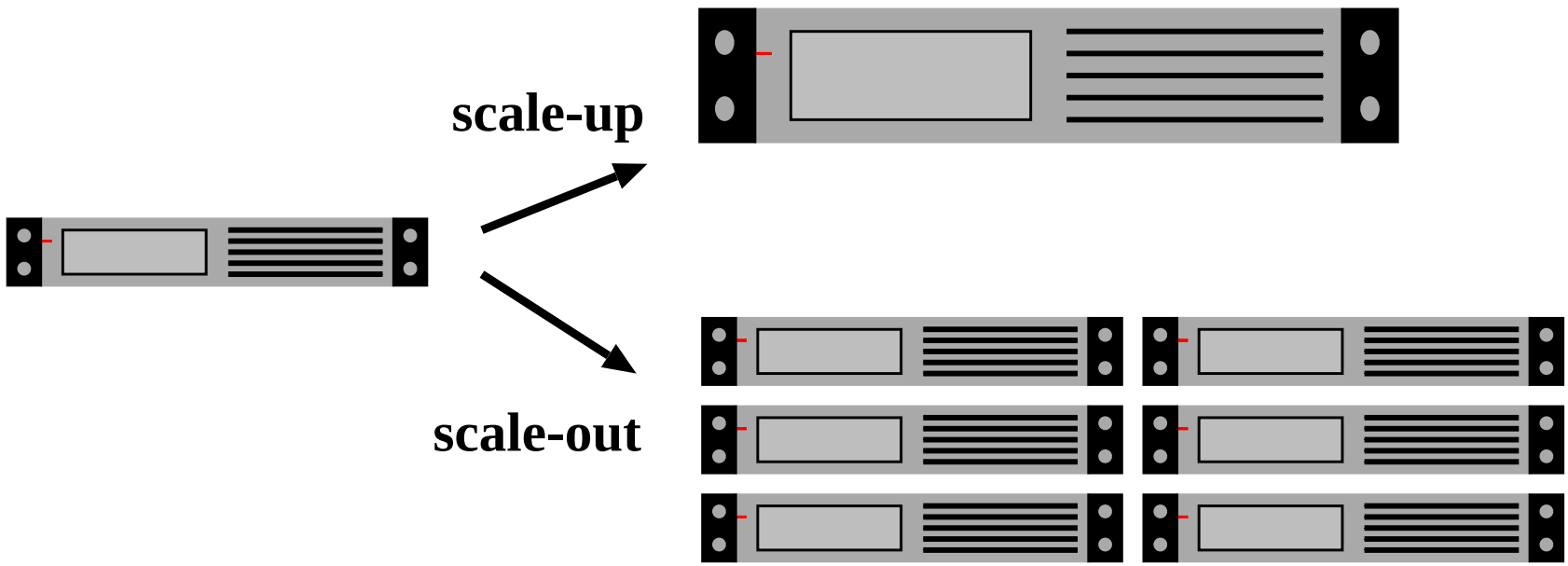
- 並列アルゴリズムのなかでも、独立したコンピュータ間のメッセージ交換のみによる通信を前提にしたもの
- コンピュータの故障やメッセージの損失を考慮

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

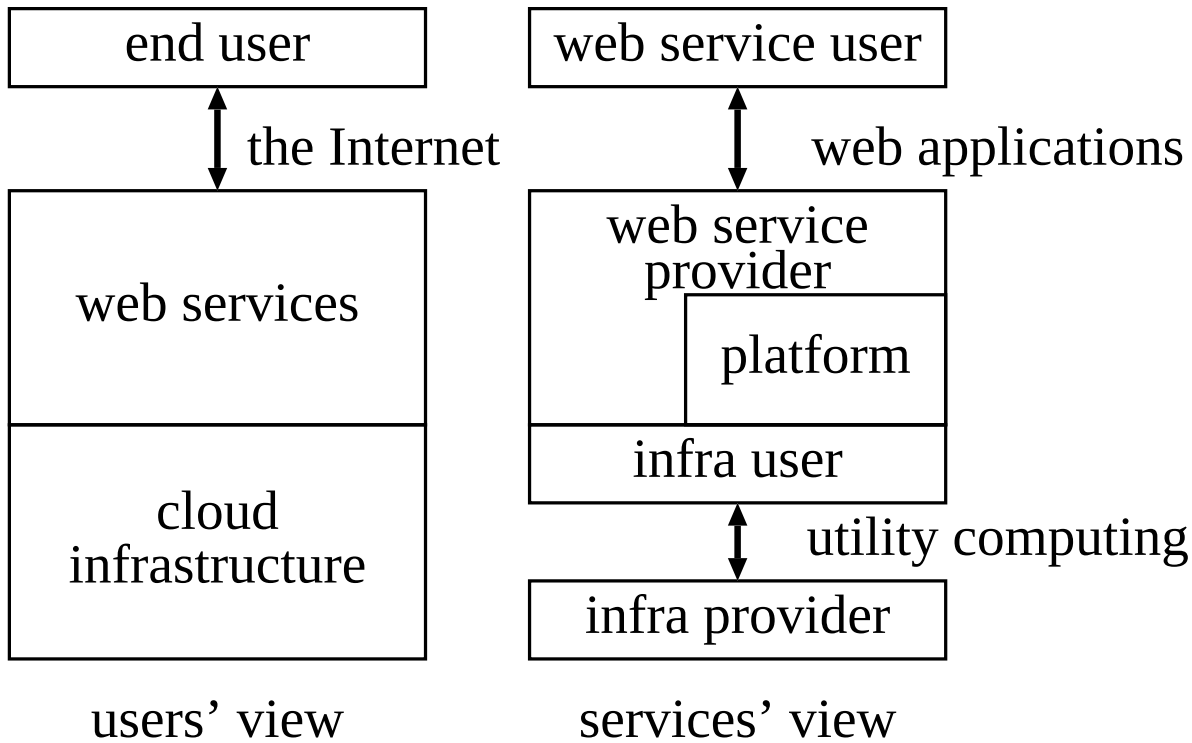
- 広くは、ネットワークの向うにあるコンピュータ資源

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



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！

クラウドコンピューティング：ネットワーク屋の視点

- 変動の大きいサービス要求
  - 多くのリソースはアイドル状態
  - 集約することで要求を平滑化
- どこかで聞いたような、、、
  - パケット交換！
  - バースト的なコンピュータ通信

しかし、VM レベルの抽象化だけでは、パラダイムシフトは起こらない

big data

- big data: 大量の非定型データから隠れた価値のある情報を引き出す技術の総称
  - 新たなビジネスモデルの構築や経営改革に繋げる
- big data という言葉をいたるところで聞くようになった
- 技術は以前から使われている
  - 検索ランキング、オンラインストアのお勧めシステムなど
  - インターネット計測: 大量かつ不完全なデータからインターネットを把握する試み
    - 統計的な手法による推測
    - 工学的な計測との対比

クラウドサービスの登場

- 以前は大量のデータの利用は、インハウスで収集、管理、分析ができる組織に限られていた
- クラウドサービスの普及で、誰でも大量データが使える環境が出来てきた
- 顧客のオンライン行動履歴を収集分析するパッケージツールも利用可能に
- 僅かな初期投資で顧客情報をマーケティング利用可能になった

データの時代

- あらゆる分野でデータ革命と呼べる技術革新が進行中
  - それまで難しかった応用が可能に
    - 膨大なデータへのアクセス、常に更新されるデータを対象にした解析、非線形モデルへの応用など
- あらゆる科学技術分野で、膨大なデータ解析は欠かせない研究手法になった

ビッグデータの技術

- データの収集
  - 利用者のオンライン行動履歴のマーケティング利用
  - センサー情報やソーシャルメディアなどあらゆる情報がオンラインに
- データの保存
  - 分散ストレージ、NoSQL、データベース
- データの処理
  - クラウドコンピューティング、MapReduce などの分散処理
- データの理解と学習
  - データマイニング、機械学習、統計処理などのツール

データ分析はあくまで道具

- 最近のビッグデータの話題はツールや手法が強調されがち
- データ解析はあくまでツール
  - 仮説を立てて、データで検証
  - 結果が予想と異なれば、そこから新たな疑問へ
  - このプロセスの繰返しから、役立つ情報や興味深い事実の発見
- 目的を持たずにデータを集め CPU を回し解析してもムダ
- 逆にデータから何を得たいかがはっきりすれば、やるべきことは見えてくる

思考プロセスの変化

- もちろん以前からデータを基に考えることは重要だった
- 情報技術によって、データに基づいて考え、考えをデータで検証する思考プロセスに変化
  - 扱えるデータの量と質、その表現方法が桁違いに
  - 文字通りデータと対話しながら考えることが可能に

データ時代の課題

- 人材 (データサイエンティスト) の育成
  - その分野の専門知識を持った上で、既存の考えや解釈に疑問を持つ、統計やデータ解析を道具として使いこなして問題解決をする
- データの財産化
  - 他社が持っていないような実データを持つ会社が強い
  - 同じデータなら、情報を引き出す能力で優劣
- データの共有
  - データを共有できる、検証できることの社会的意義
- プライバシーとのバランス: 社会的合意形成が大きな課題
  - 組織がどこまで個人を追跡していいか
  - 個人の医療情報などをどのように共有して社会に役立てるか

受け取りてのリテラシ

- 受け取り側も、データを理解する、データに疑問を持つ必要
  - 発信者のバイアスによる作為的な統計データや情報操作の氾濫
- 我々は白黒の判定を求めがち
  - ほとんどの物事はグレー、白黒は便宜的にグレーに線を引く行為
  - 白黒を求めるのは、自ら判断することを避けて、発信者に判断の責任を求める行為
  - グレーはグレーとして受け取り、自分で判断することが必要な社会になってきている

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)) → (1 4 9 16)
(reduce + '(1 4 9 16)) → 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

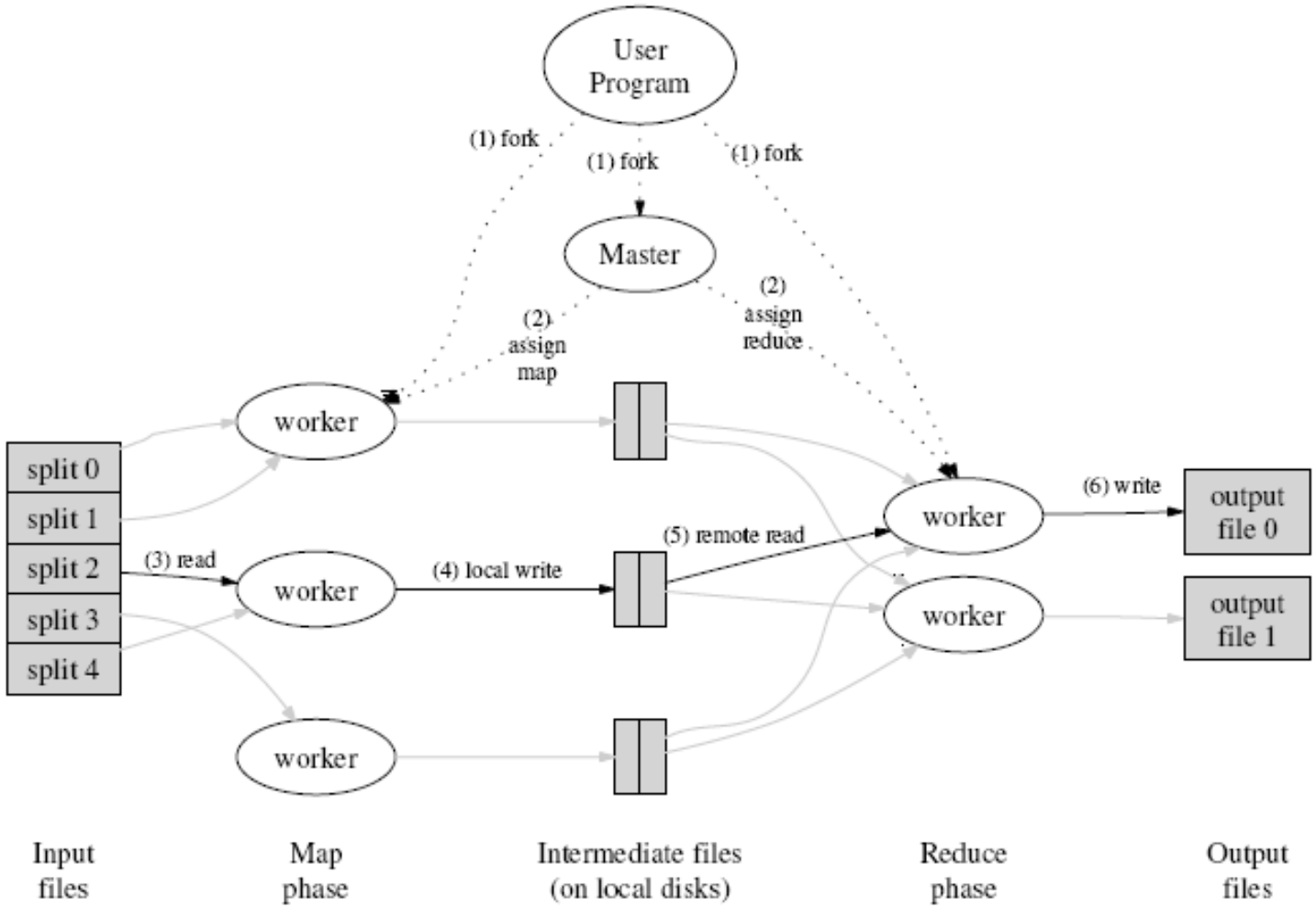
```
map(String input_key, String input_value):
  // input_key: document name
  // input_value: document contents
  for each word w in input_value:
    EmitIntermediate(w, "1");

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

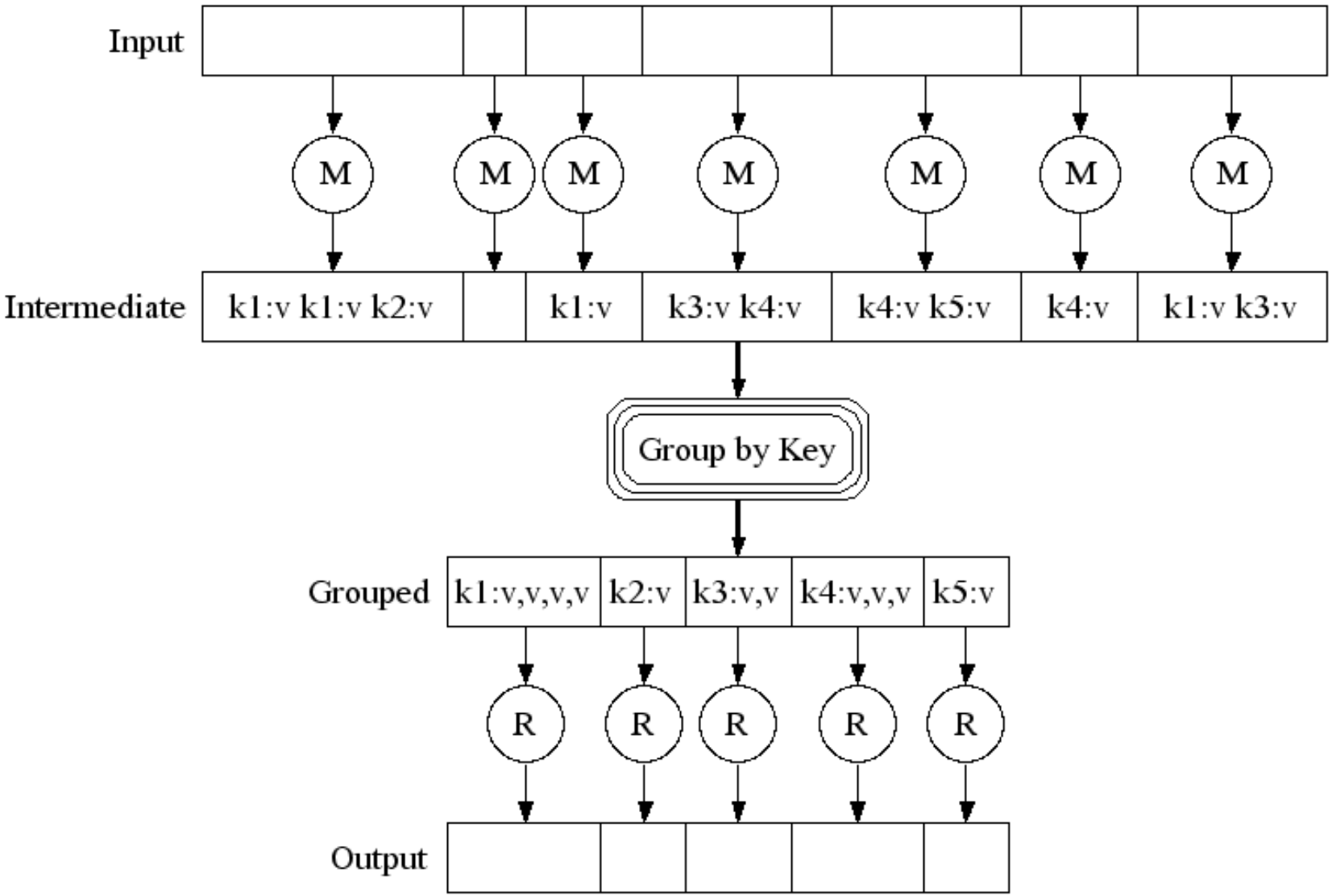
- 分散 grep
  - map: 特定パターンにマッチする行を出力
  - reduce: 何もしない
- URL アクセス頻度カウント
  - map: web access log から $< URL, 1 >$ を出力
  - reduce: 同一 URL の回数を加算し $< URL, count >$ を生成
- reverse web-link graph
  - map: source に含まれる link から、$< target, source >$ を出力
  - reduce: target を link する source list $< target, list(source) >$ 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



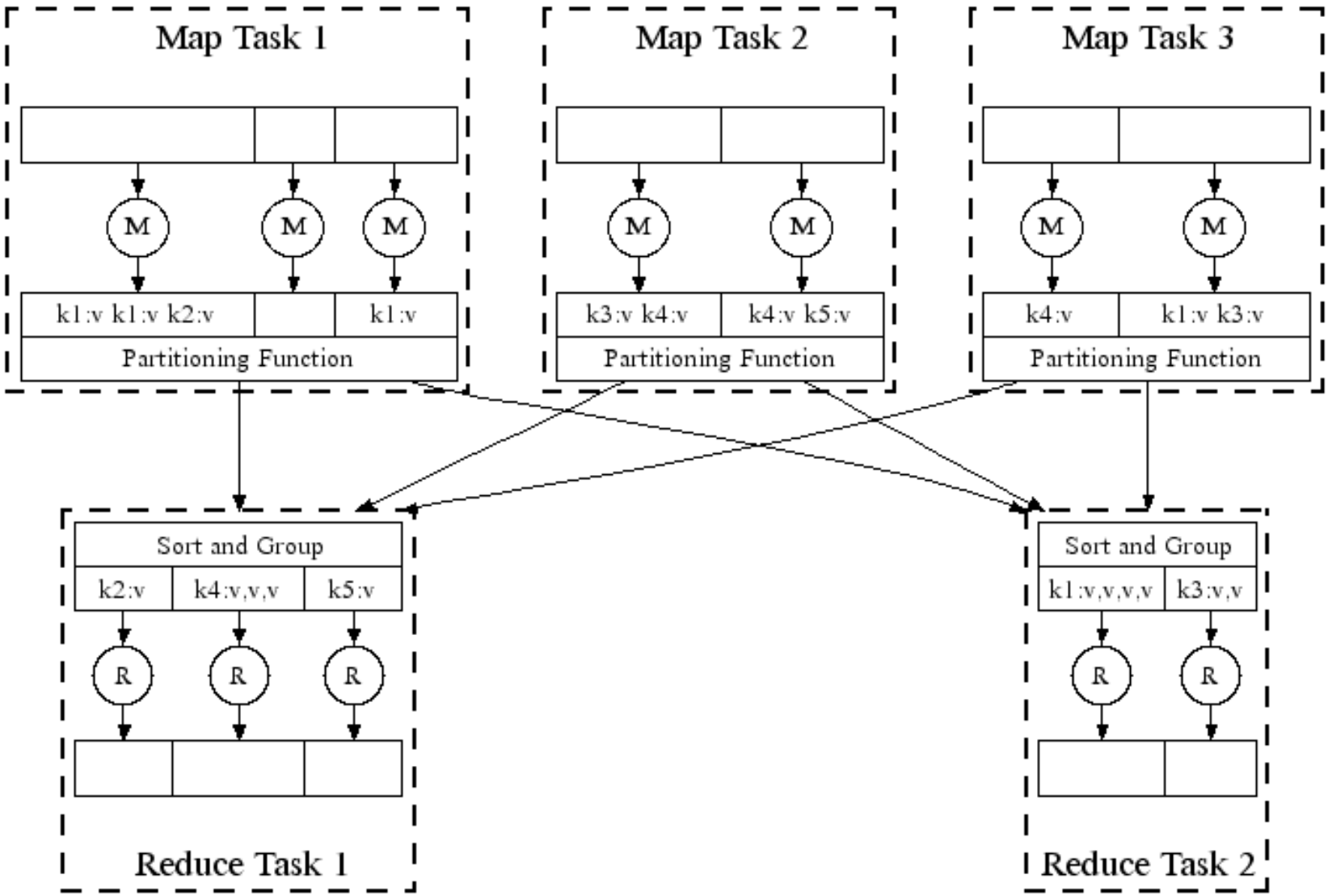
source: MapReduce: Simplified Data Processing on Large Clusters

MapReduce Execution



source: MapReduce: Simplified Data Processing on Large Clusters

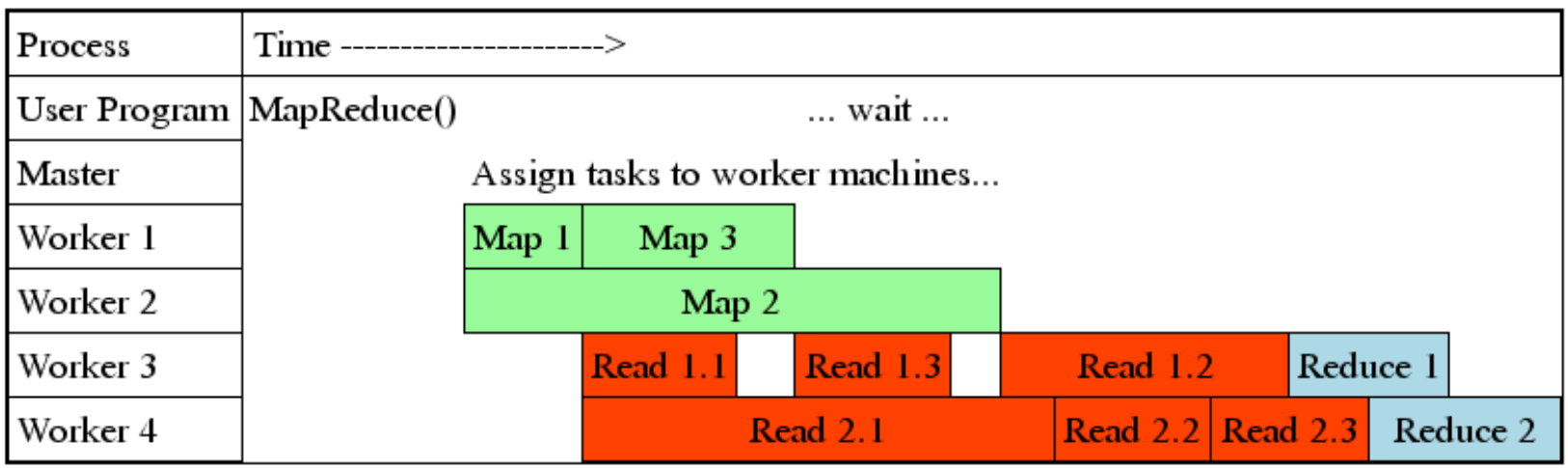
MapReduce Parallel Execution



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

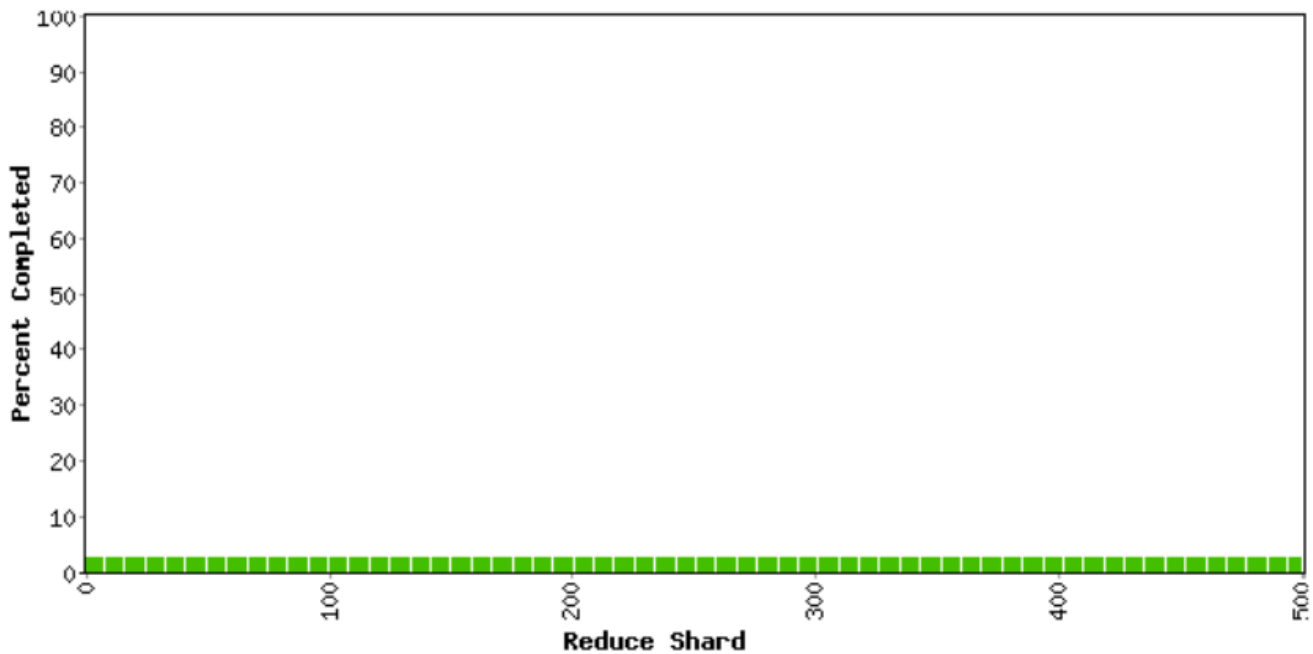
MapReduce status

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

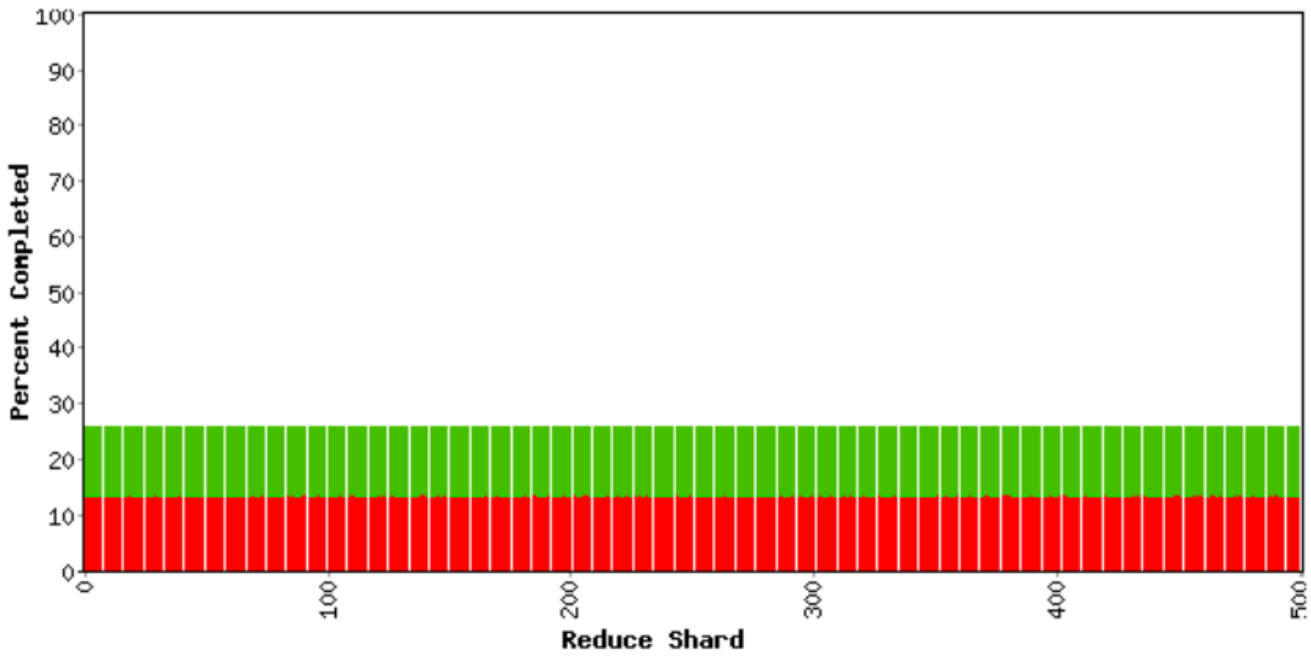
MapReduce status

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

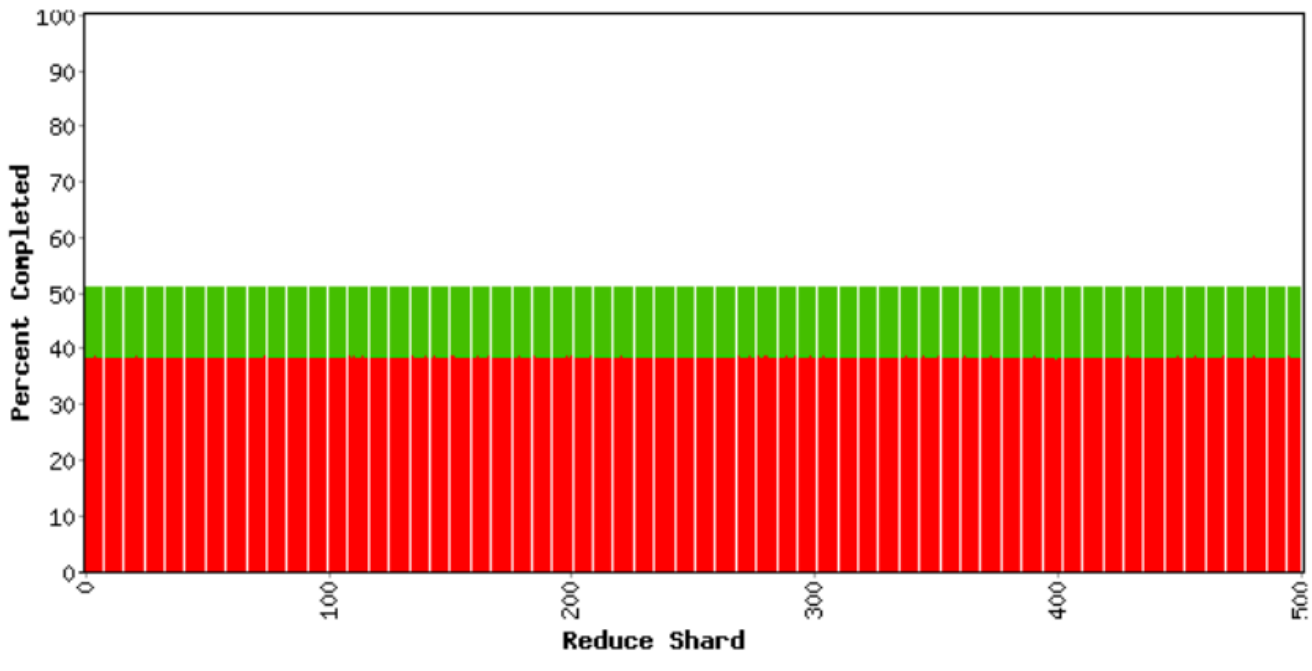
MapReduce status

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

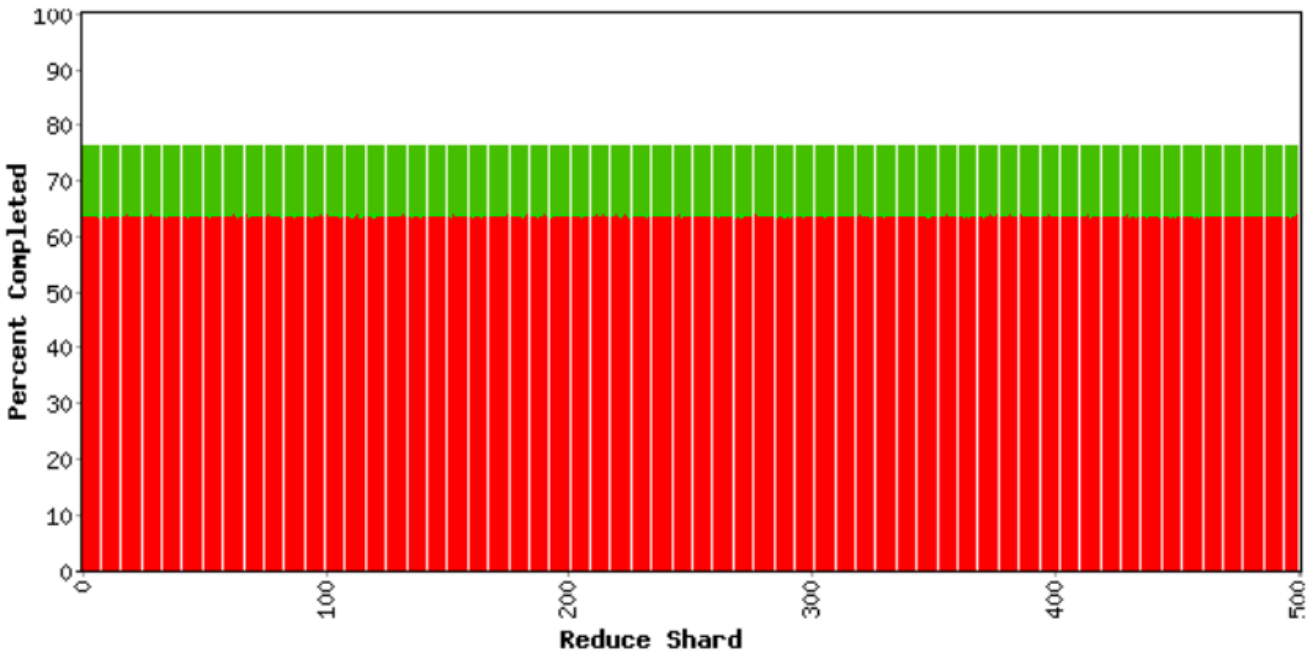
MapReduce status

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

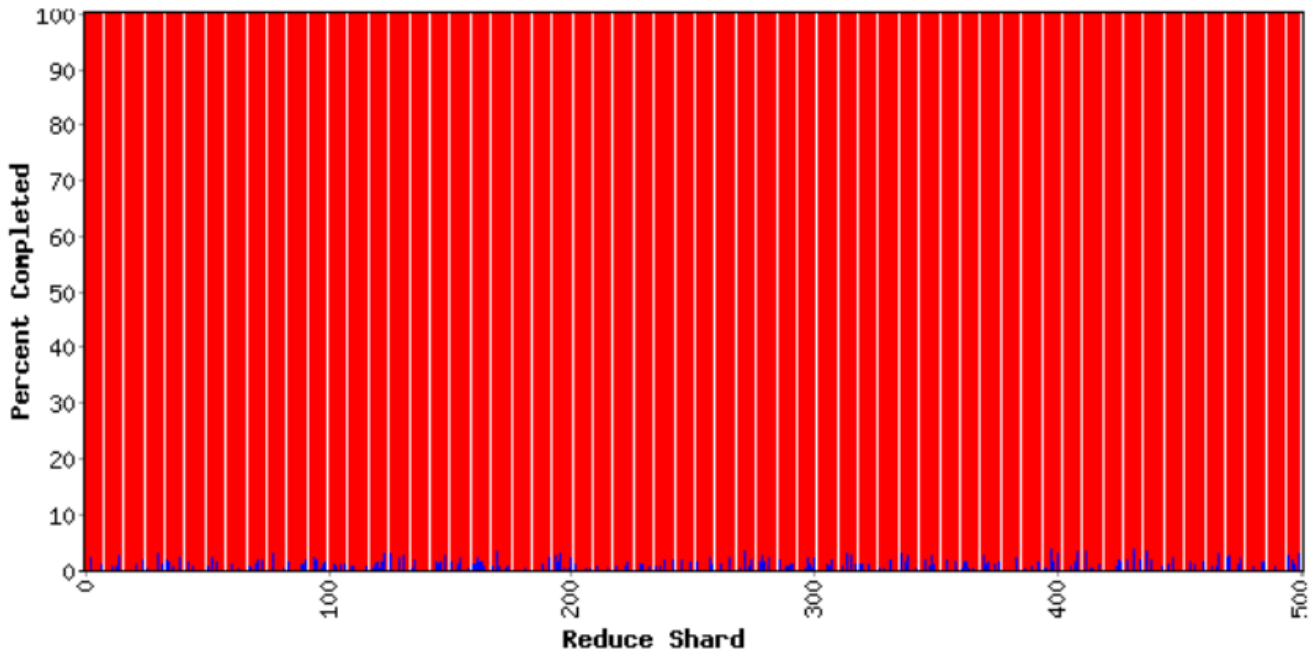
MapReduce status

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

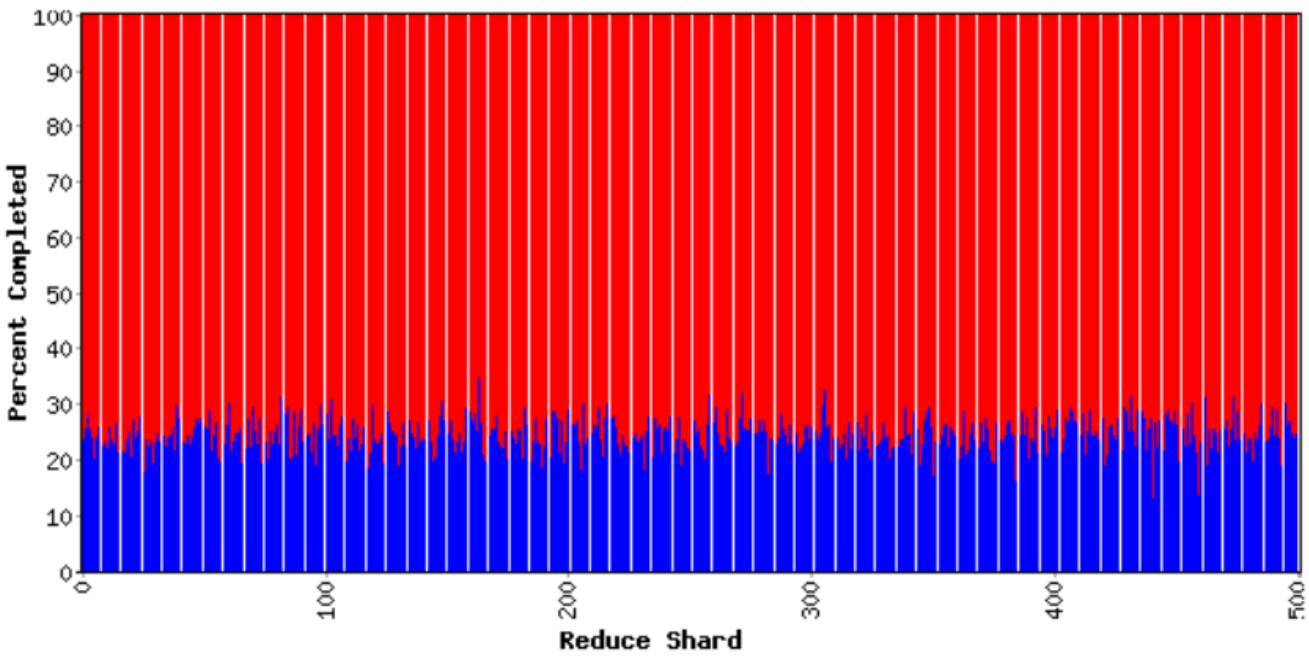
MapReduce status

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

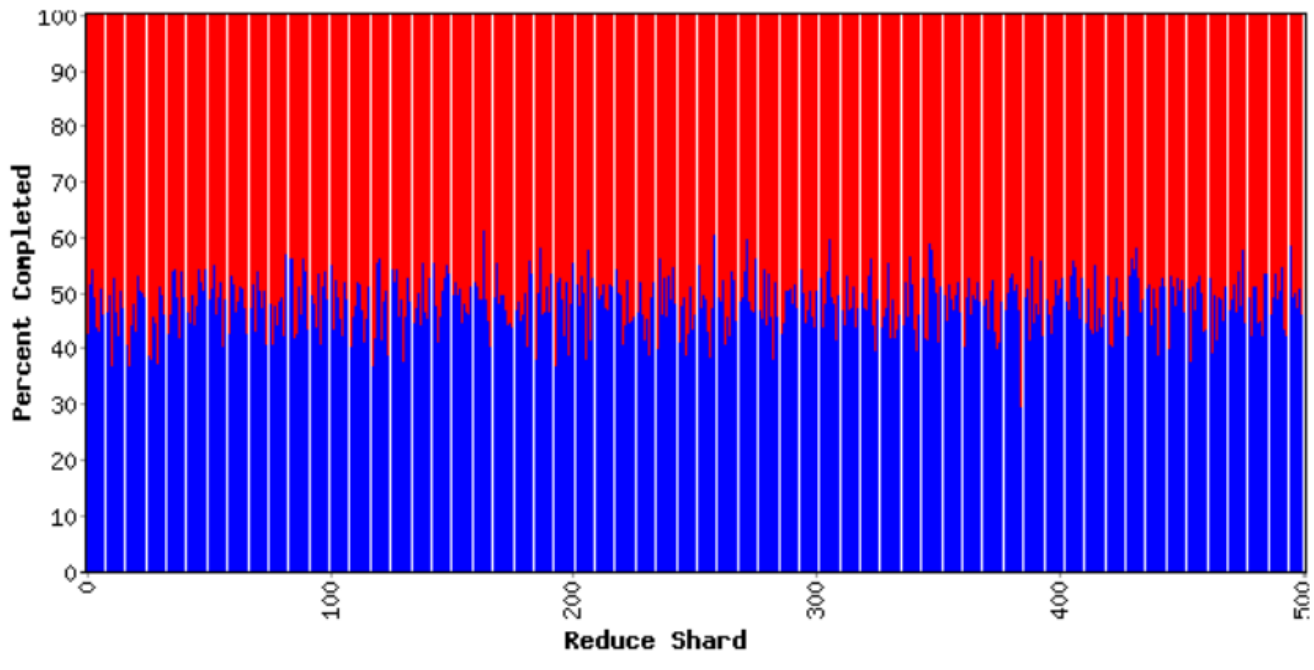
MapReduce status

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
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	51842100
mr-merge-outputs	51842100

source: MapReduce: Simplified Data Processing on Large Clusters

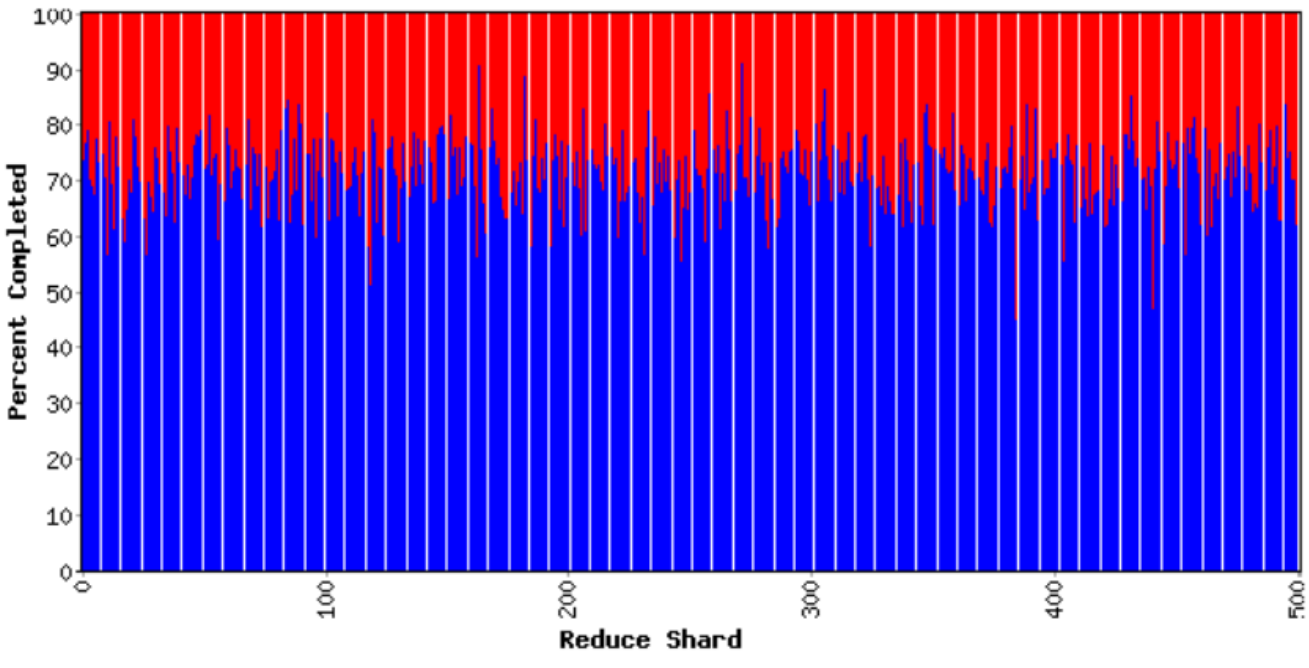
MapReduce status

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

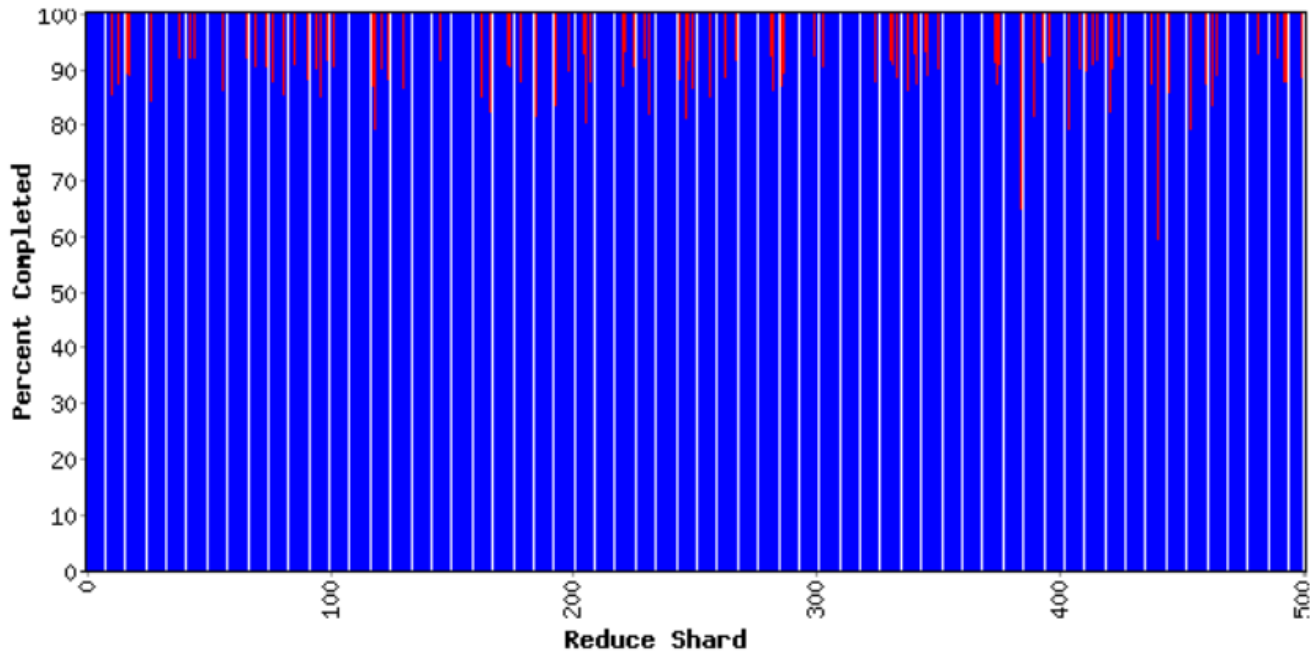
MapReduce status

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

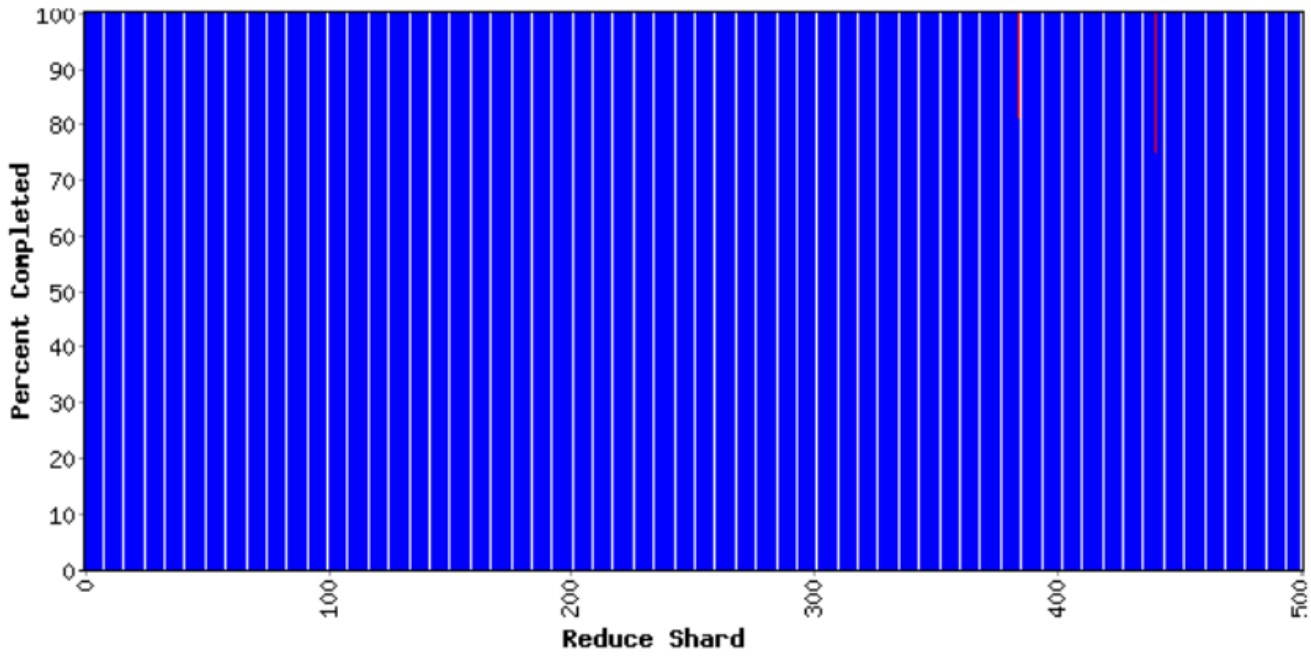
MapReduce status

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	105
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	394792	
mr-merge-outputs	394792	

source: MapReduce: Simplified Data Processing on Large Clusters

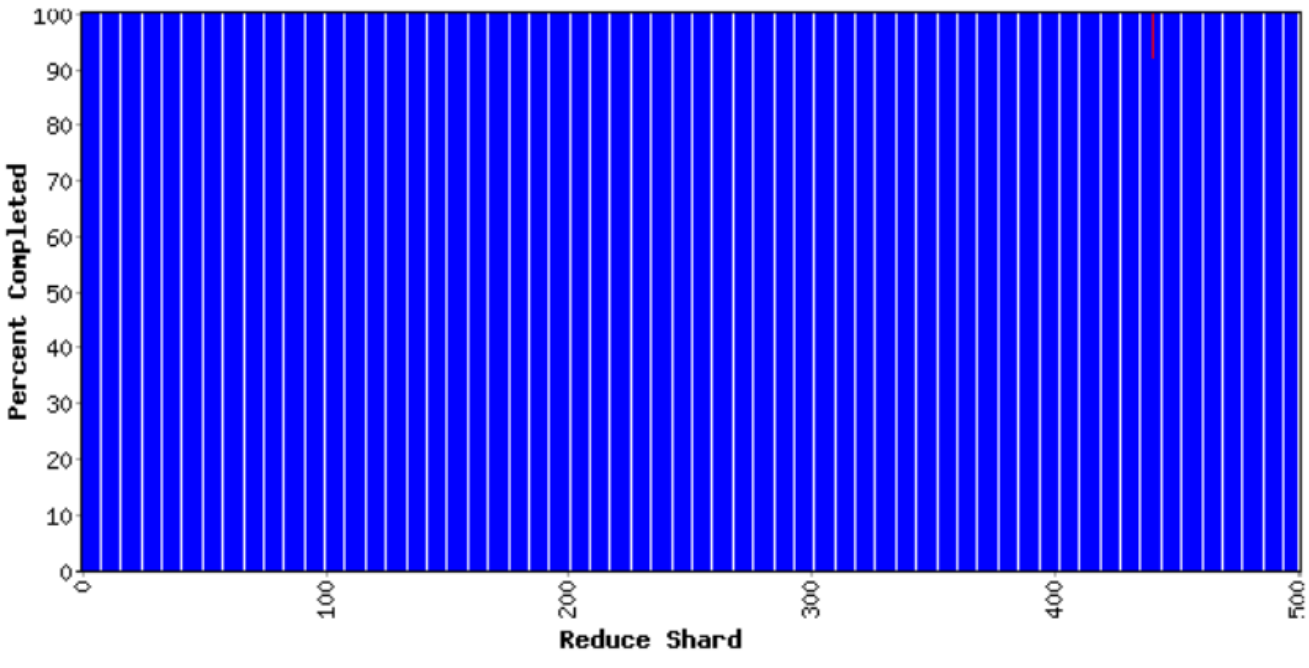
MapReduce status

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

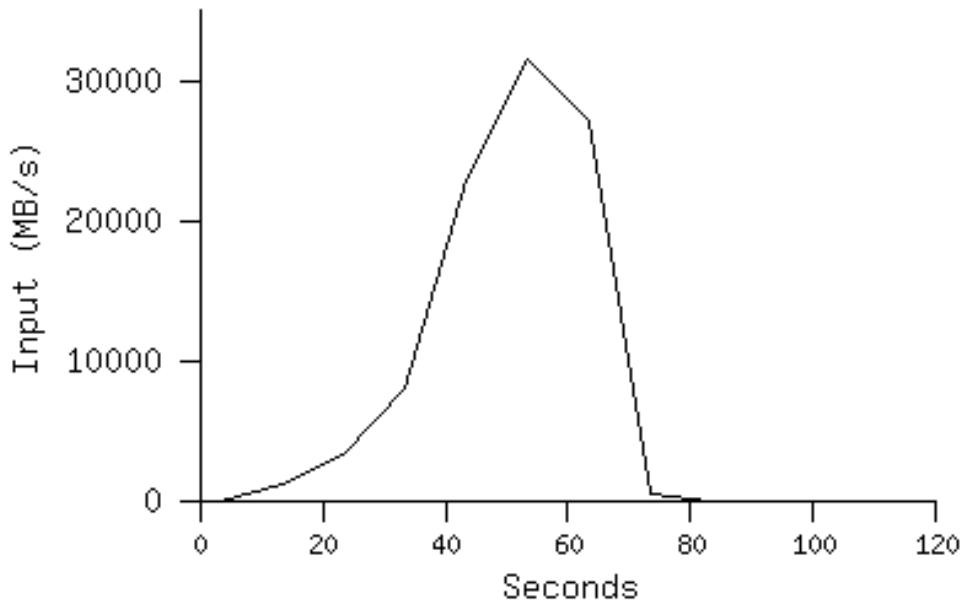
- 4GB of memory
- Dual-processor 2GHz Xeons with Hyperthreading
- Dual 160GB IDE disks
- Gigabit Ethernet per machine
- Bisection bandwidth approximately 100Gbps

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

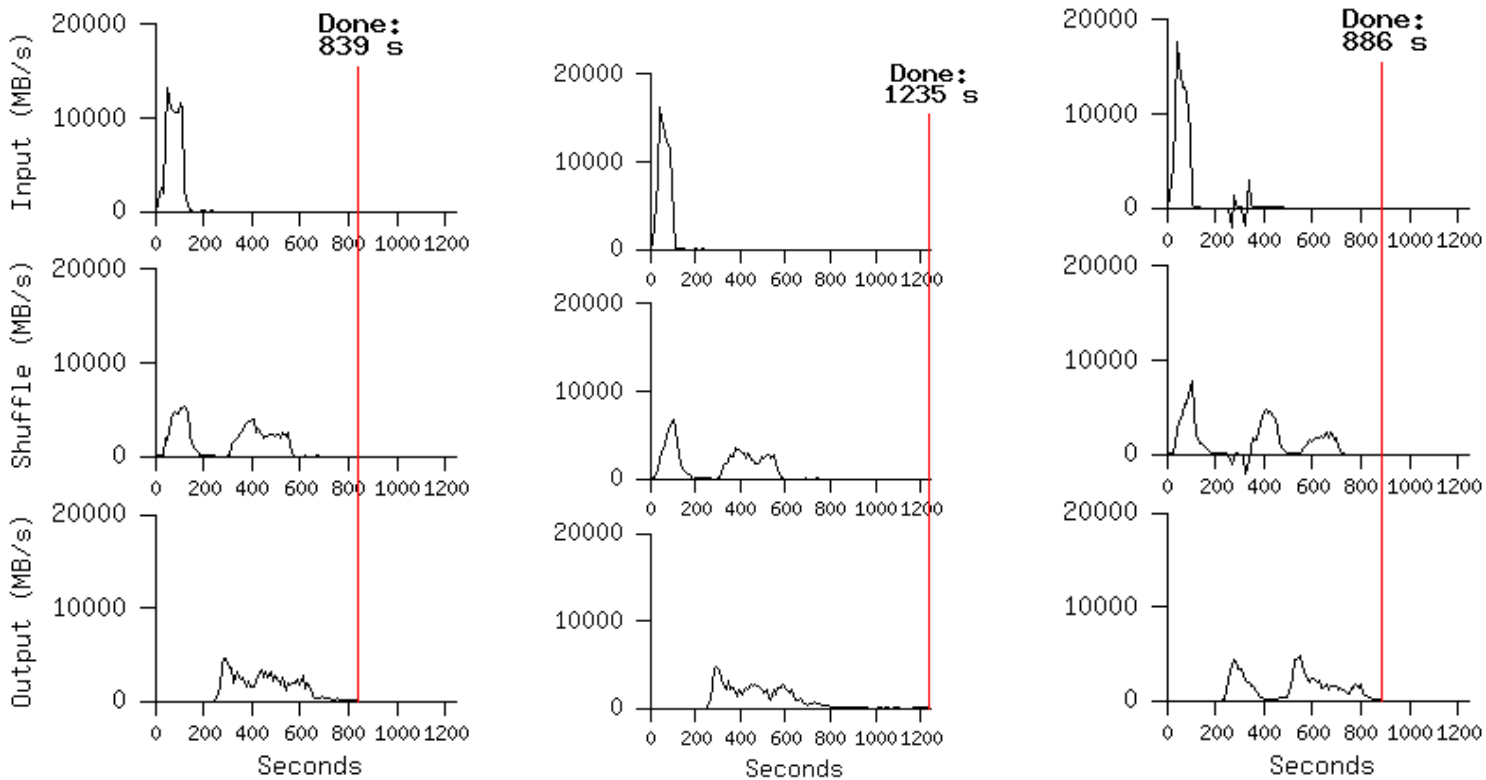
- ローカリティ最適化効果
  - 1800 台のマシンが 1TB のデータを最大 31GB/s で読み込み
  - さもなければ、ラックスイッチがボトルネックで 10GB/s 程度
- 短いジョブでは始動時のオーバヘッドが大きい



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

インターネット計測とプライバシー

計測はすべての技術の基本
計測情報の開示: 個人情報を含まない統計情報のみ開示可能
計測データからプライバシー情報が漏洩するリスク

- 計測データ中のプライバシー情報 (IP アドレスなど)
- 技術の進歩で情報の拡散や加工が容易になった
- 悪意の利用やリバースエンジニアリングの可能性

技術に法制度がついていけない現状

- ほとんどがインターネット以前に作られた制度
- 計測には法的にはグレーな部分が多い
  - 計測に対する立場の違い、技術者の認識にも大きな温度差

通信の秘密

憲法上の通信の秘密

- 政府など公権力に対する義務

電気通信事業法第 4 条第 1 項で通信の秘密

- 電気通信事業者の取扱中に係る通信の秘密は、侵してはならない

例外

- 当事者の同意がある場合
  - ウイルスチェックサービスや迷惑メールフィルタリングサービス
- 違法性阻却事由が存在し、違法とはされない場合
  - 業務上必要な正当業務行為に当たる場合
    - 例: パケット配送のためにヘッダ情報を見る
  - 緊急避難に該当する場合
    - 例: 他のサービスに支障が出ないよう対策をする

個人情報

個人を識別することができる情報

- 氏名、性別、生年月日、住所、電話番号、家族構成、職業、年収、生体情報
- IP アドレス、メールアドレス、オンライン上の ID、位置情報
- 日本の個人情報保護法　2005 年に施行
  - 5000 件以上の個人情報を扱う事業者が対象
  - 利用目的の特定、制限、適切な取得、通知義務、苦情処理

プライバシー

みだりに自分の私生活を公開されない権利、法的保証
個人の情報を自分でコントロールできる権利
プライバシー情報

- 利用したサービス、web 閲覧履歴、検索履歴、購入商品、趣味指向
- 本人が自ら公開している場合はプライバシー情報とはならない
- しかし、情報の収集、加工、第三者への提供などもプライバシーの侵害になりえる

インターネット計測とプライバシー漏洩リスク

生データ、汎用データ

- 当初の目的以外の利用が可能、いっぽうで情報漏洩リスクを伴う
- 汎用性と情報漏洩リスクのトレードオフ
  - 例えば、特定目的用にオンライン処理することでリスク減少

データの共有、公開

- 共有: 第三者への情報提供となる問題
  - 必要最小限の情報のみ共有するようなデータの加工は可能
- 公開: 幅広い利用促進、悪用されるリスク

商用トラフィックと非商用トラフィック

- 研究教育用ネットワークは比較的計測しやすい
- いっぽうで、商用トラフィックとの乖離

インフォームド コンセント

- 利用者に説明、理解と合意を得るプロセス
- 医療分野で進んでいる (倫理委員会設置など)

法的側面とモラル

- 合法であるかだけでなく、技術者のモラルが問われる
  - センシティブなデータの削除や匿名化

まとめ

スケールする計測と解析

- 大規模計測
- MapReduce
- 分散並列処理
- クラウド技術
- インターネット計測とプライバシー
- 演習: MapReduce