# Introduction to MapReduce Algorithms and Analysis

Jeff M. Phillips

October 25, 2013

# Trade-Offs

*Massive parallelism that is very easy to program.*

- Cheaper than HPC style (uses top of the line everything)
- Assumption about data (key-value pairs).
- Restrictions on how computation is done.

# Cluster of Commodity Nodes (2003)

Big Iron Box:

- 8 2GHz Xeons (Processors)
- 64 GB RAM
- 8 TB disk
- 758,000 USD

Google Rack:

- 176 2GHz Xeons (Processors)
- 176 GB RAM
- 7 TB disk
- 278,000 USD

# Google File System

SOSP 2003 (Ghemawat, Gobioff, Leung)

**Key-Value Pairs:**
All files stored as Key-Value pairs

- **key:** log id **value:** actual log
- **key:** web address **value:** html and/or outgoing links
- **key:** document id in set **value:** list of words
- **key:** word in corpus **value:** how often it appears

**Blocking:**
All files broken into blocks (often **64 MB**)
Each block has replication factor (say **3** times), stored in separate nodes.
No locality on compute nodes, no neighbors or replicas on same node (but often same rack).

# No locality?

Really?

# No locality?

Really?

- Resiliency: if one dies, use another
  (on big clusters happens all the time)
- Redundancy: If one is slow, use another
  (...curse of last reducer)
- Heterogeneity: Format quite flexible, 64MB often still enough
  (recall: IO-Efficiency)

# MapReduce

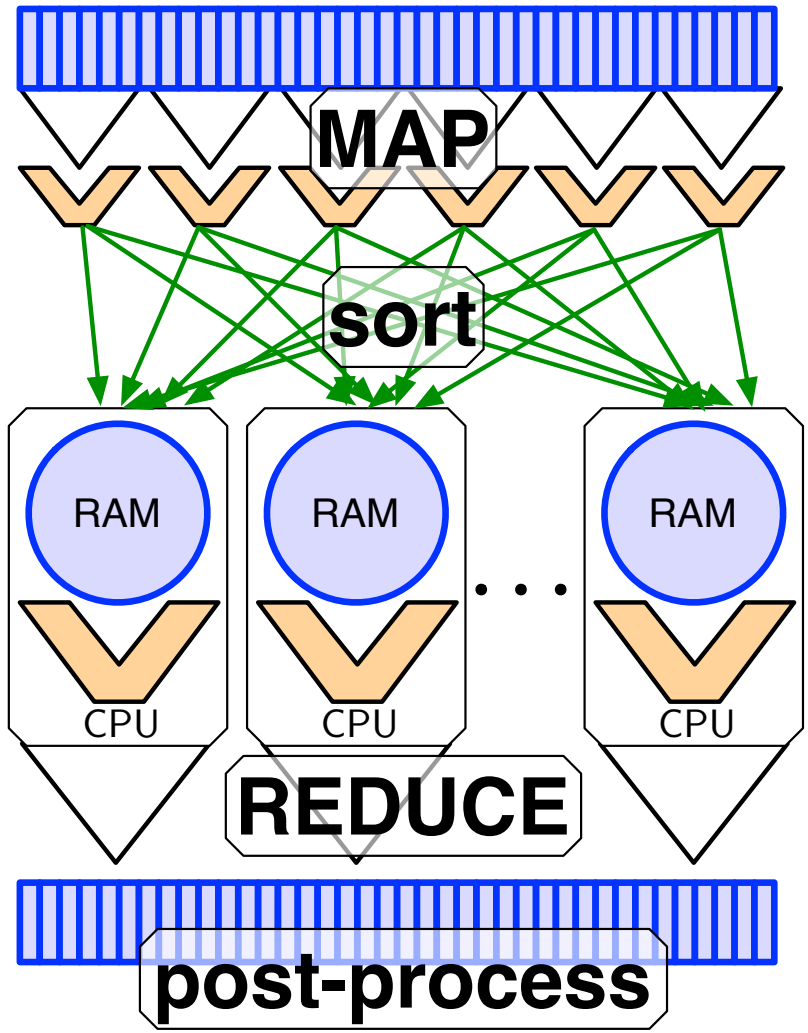
OSDI 04 (Dean, Ghemawat)

Each Processor has full hard drive, data items $< \text{KEY}, \text{VALUE} >$.
Parallelism Procedes in Rounds:

- Map: assigns items to processor by KEY.
- Reduce: processes all items using VALUE. Usually combines many items with same KEY.

**Repeat** M+R a constant number of times, often only one round.

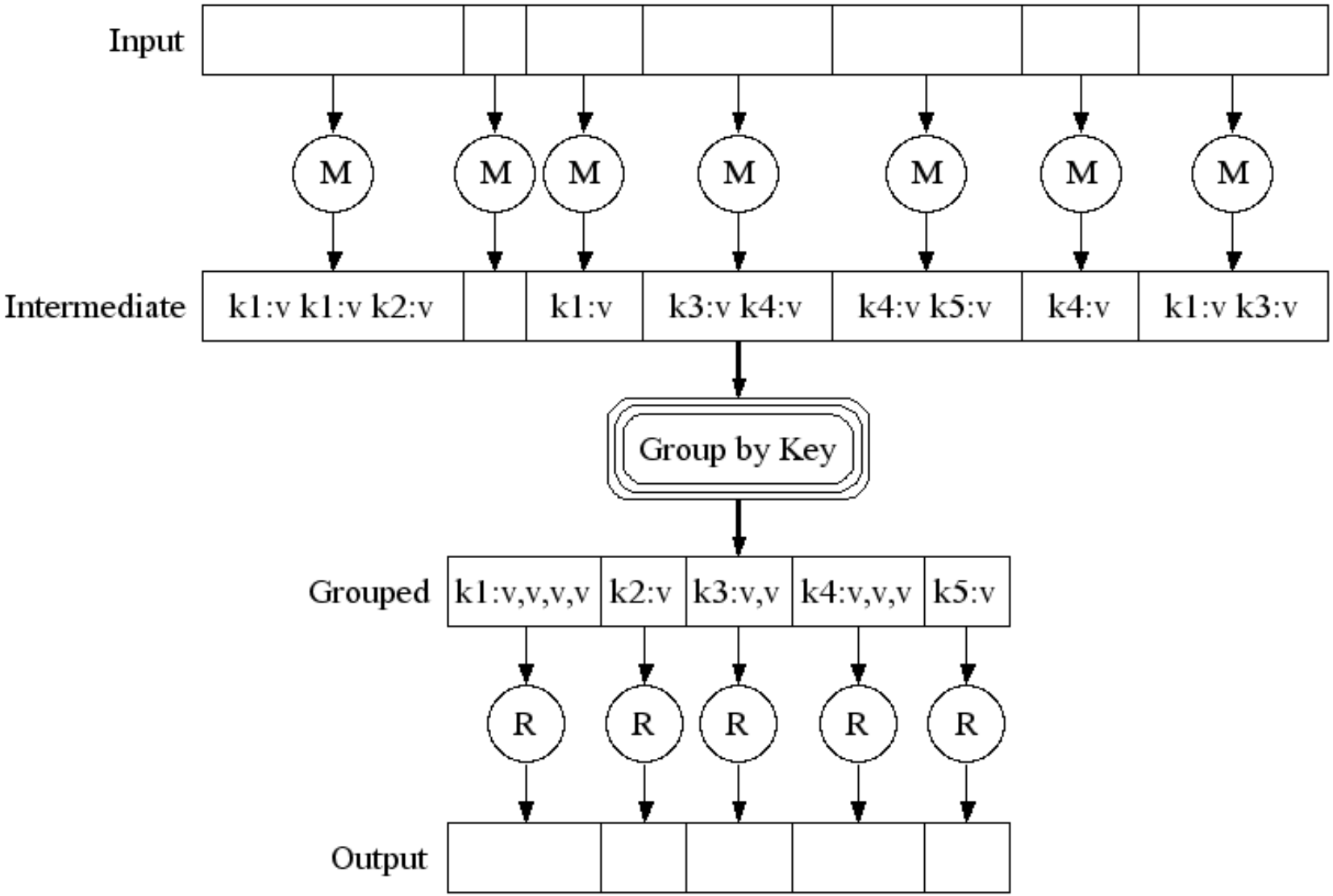
- Optional post-processing step.




Pro: Robust (duplication) and simple. Can harness Locality
Con: Somewhat restrictive model

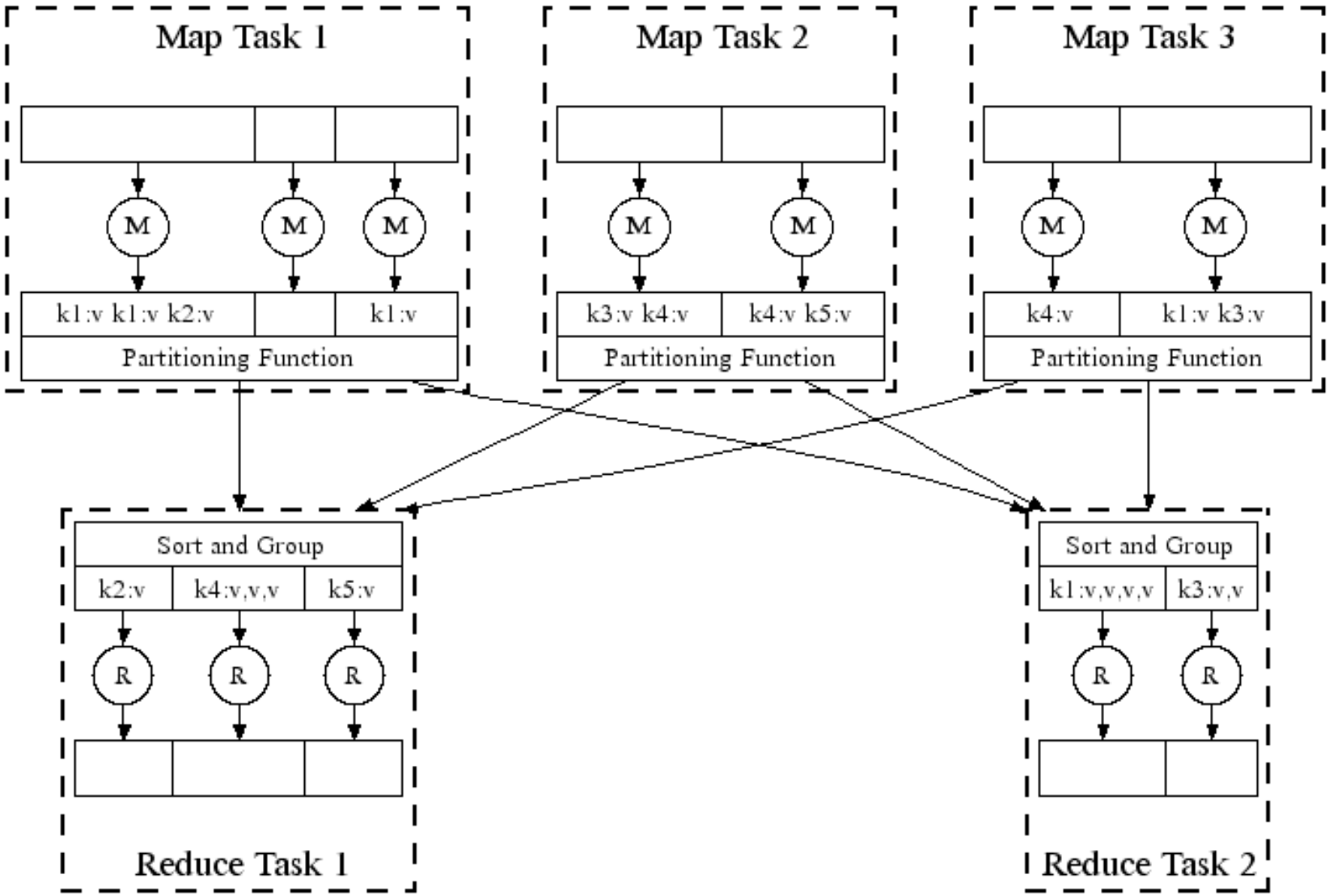
# MapReduce

OSDI 04 (Dean, Ghemawat)



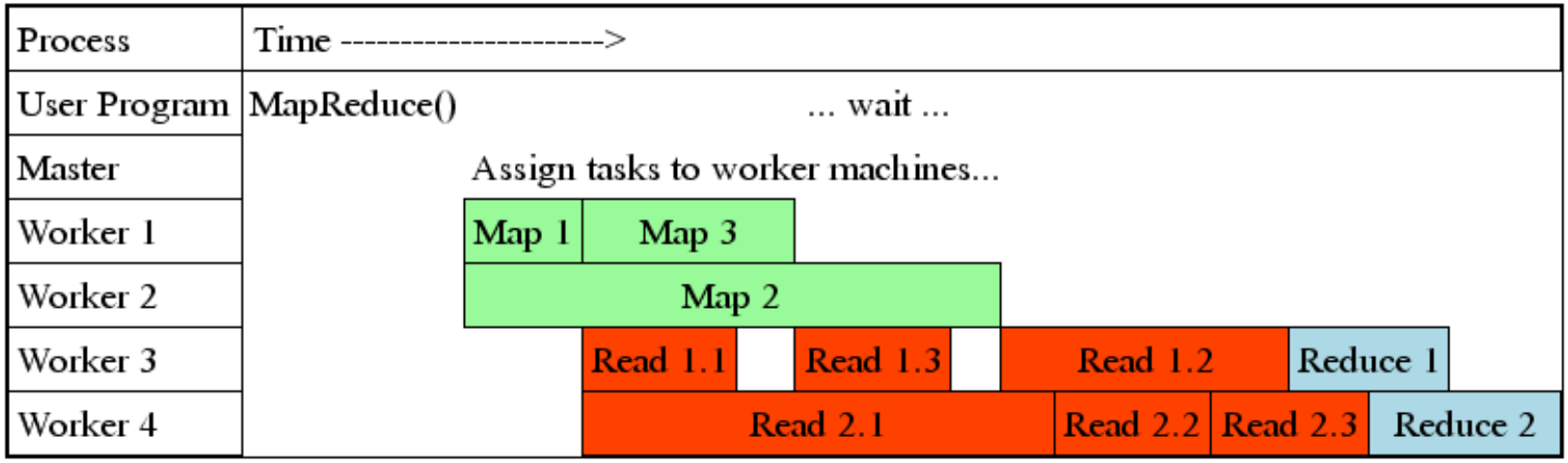
# MapReduce

OSDI 04 (Dean, Ghemawat)



# Granularity and Pipelining

OSDI 04 (Dean, Ghemawat)



# MapReduce

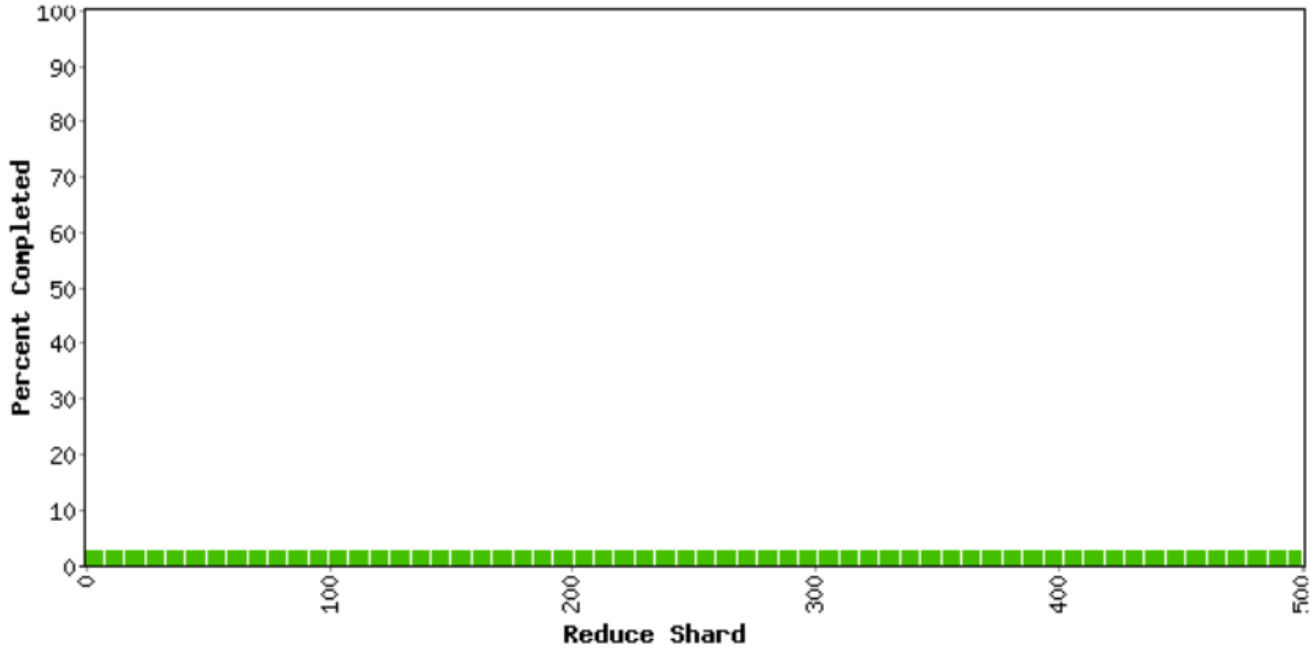
## OSDI 04 (Dean, Ghemawat)

### 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 |

# MapReduce

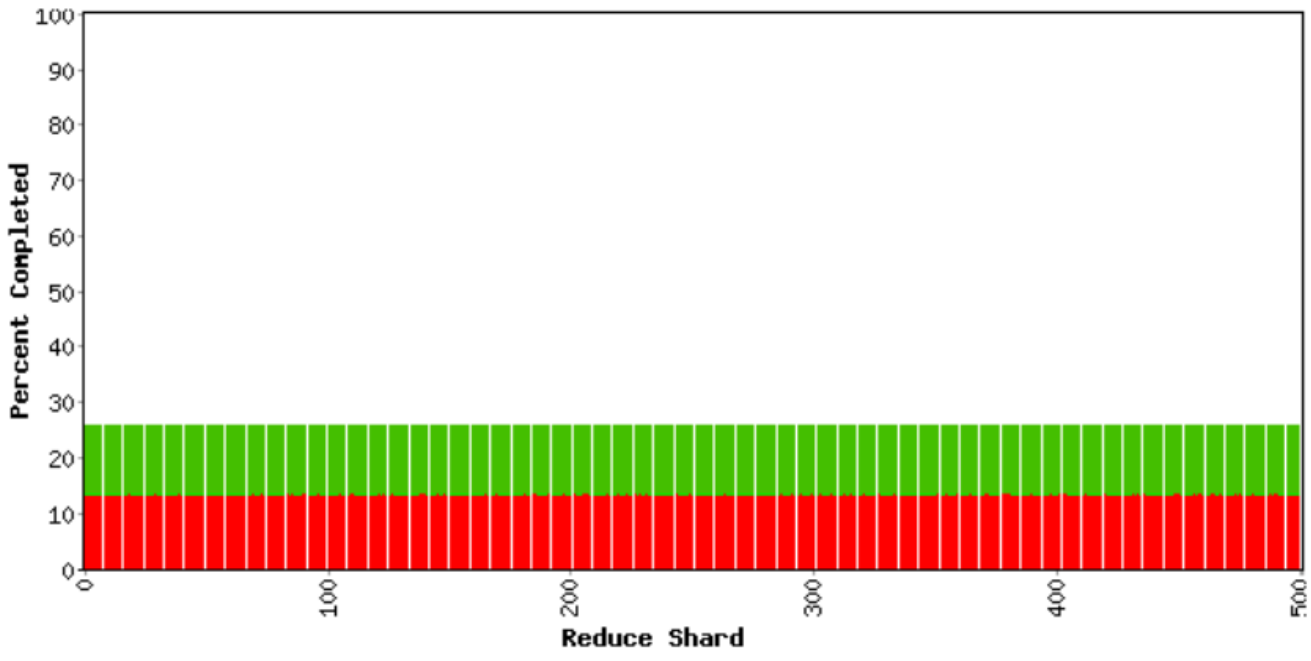
## OSDI 04 (Dean, Ghemawat)

### 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 |

# MapReduce

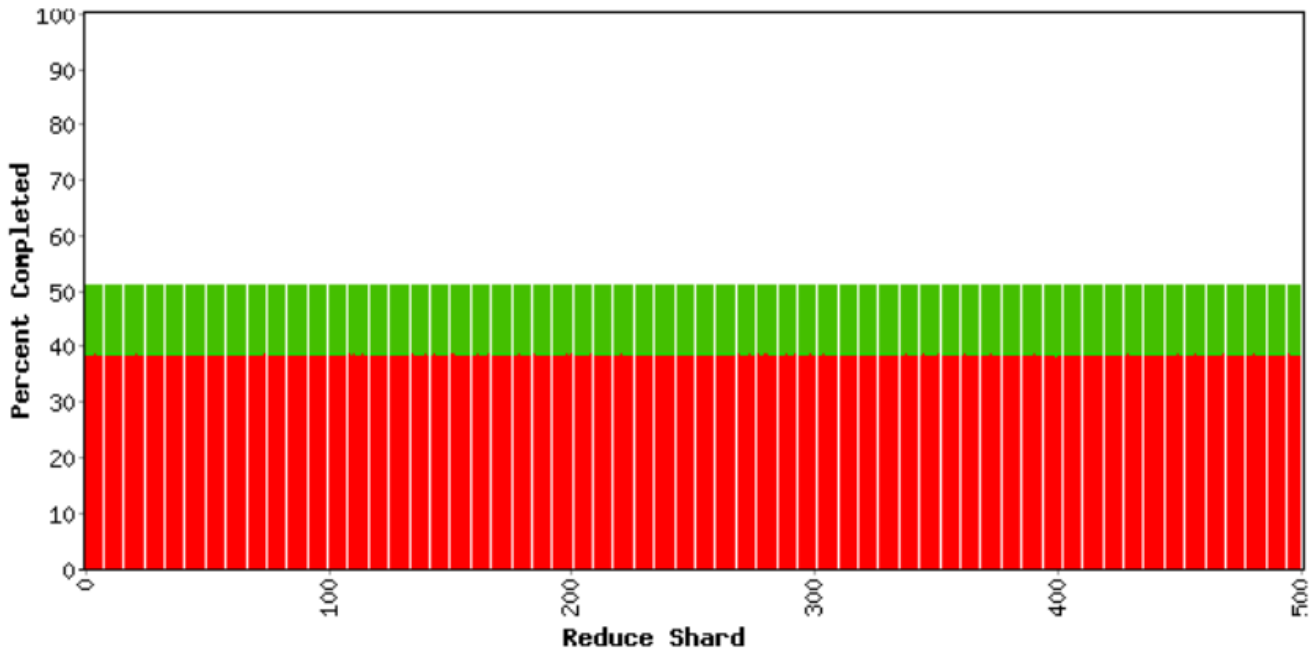
## OSDI 04 (Dean, Ghemawat)

### 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 |

# MapReduce

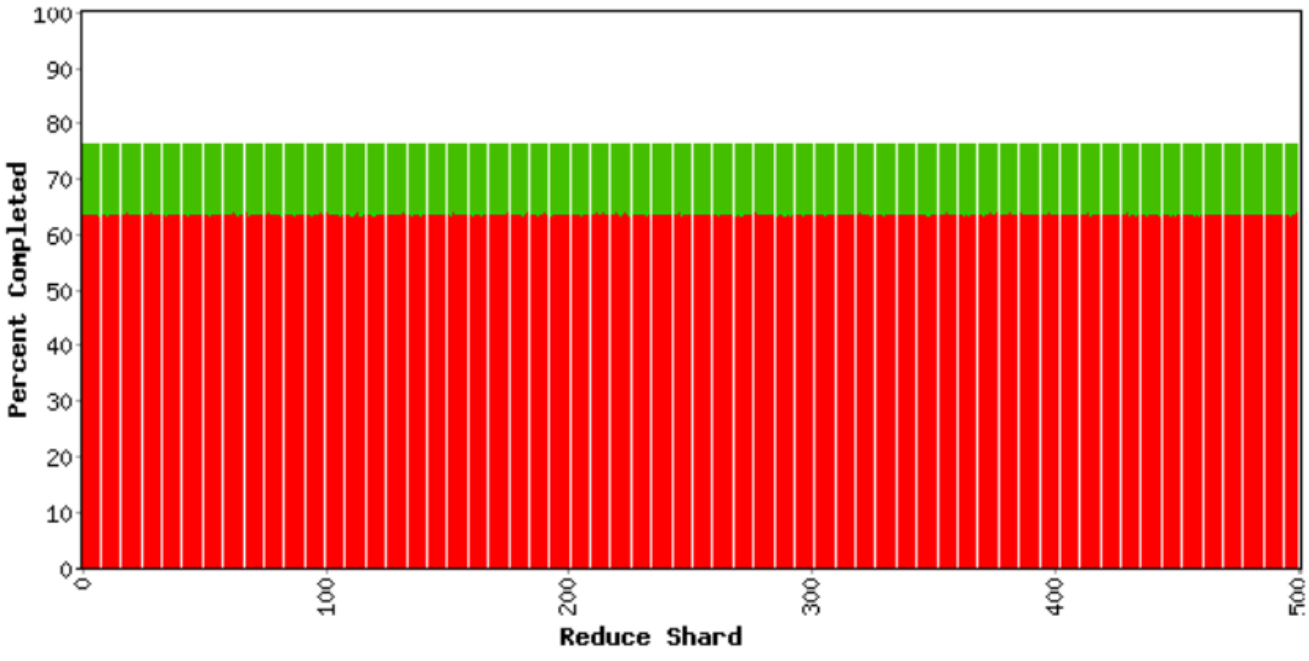
## OSDI 04 (Dean, Ghemawat)

### 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 |

# MapReduce

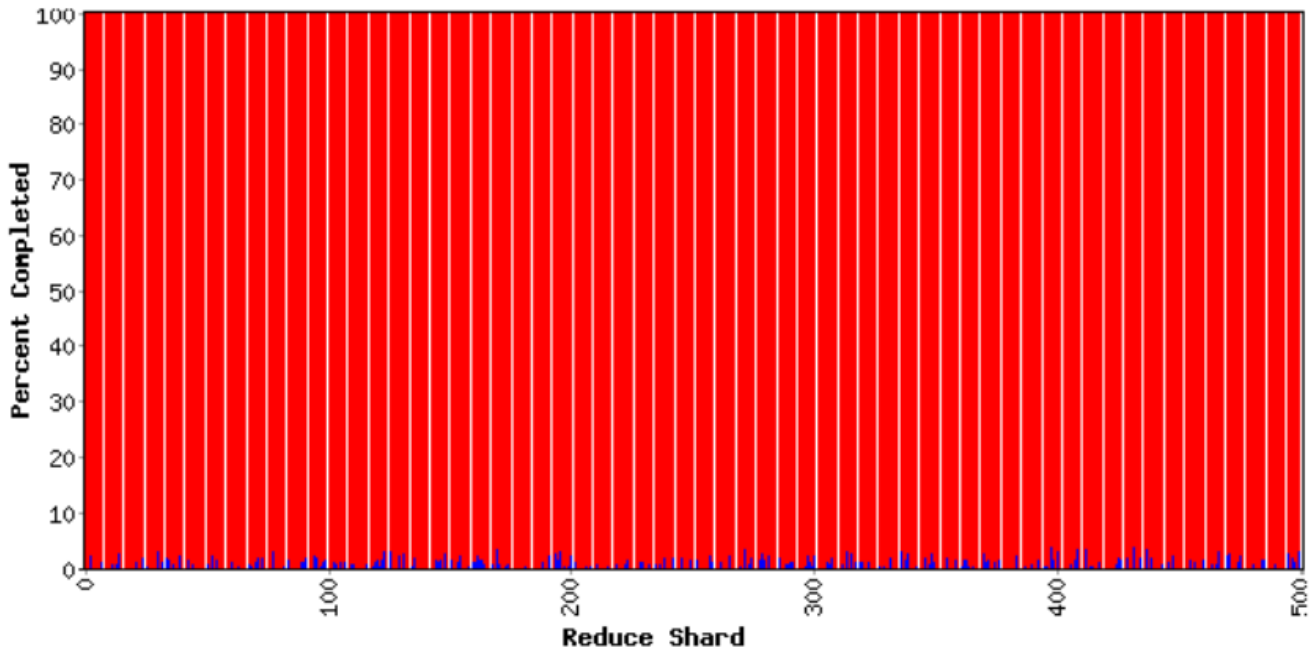
## OSDI 04 (Dean, Ghemawat)

### 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 | |

# MapReduce

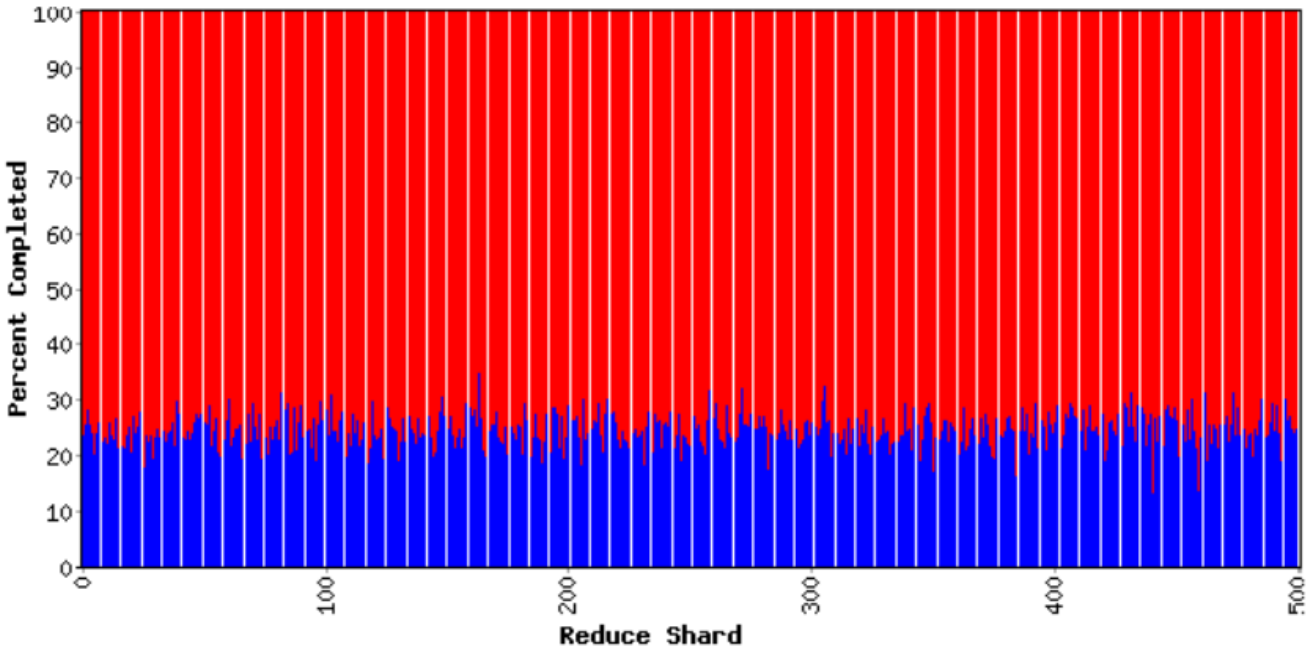
## OSDI 04 (Dean, Ghemawat)

### 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 |
| docs-indexed | 0 |
| dups-in-index-merge | 0 |
| mr-merge-calls | 51738599 |
| mr-merge-outputs | 51738599 |

# MapReduce

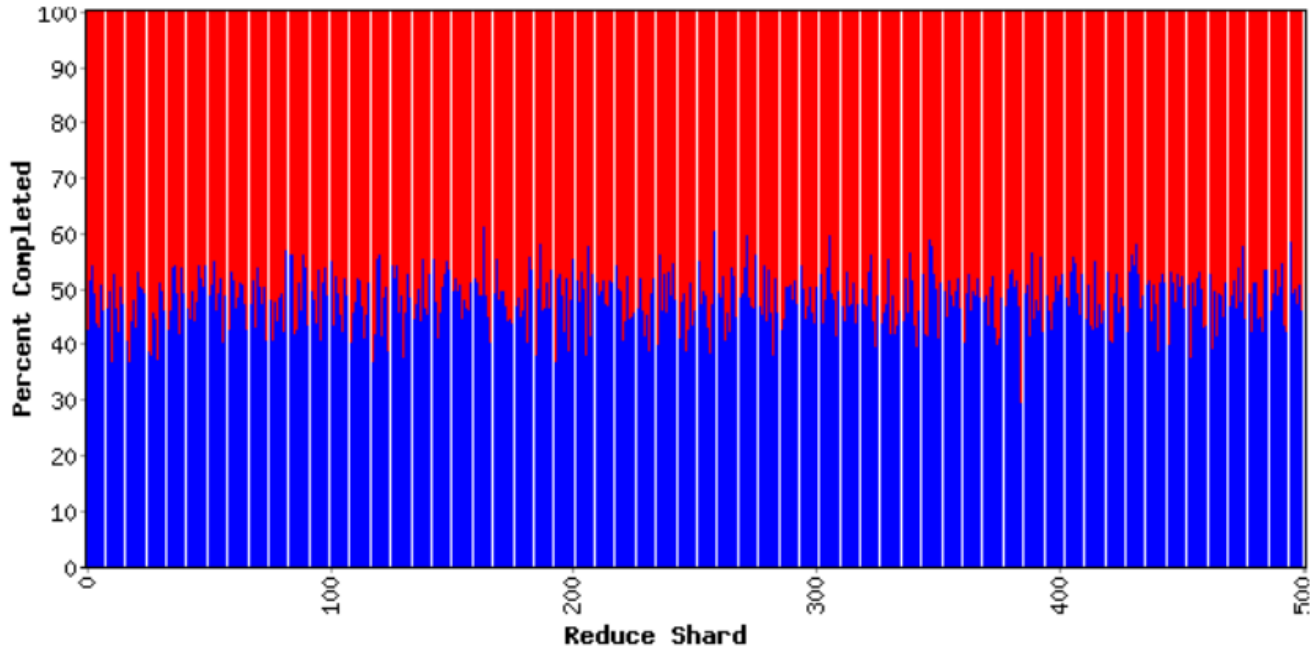
## OSDI 04 (Dean, Ghemawat)

### 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 |

# MapReduce

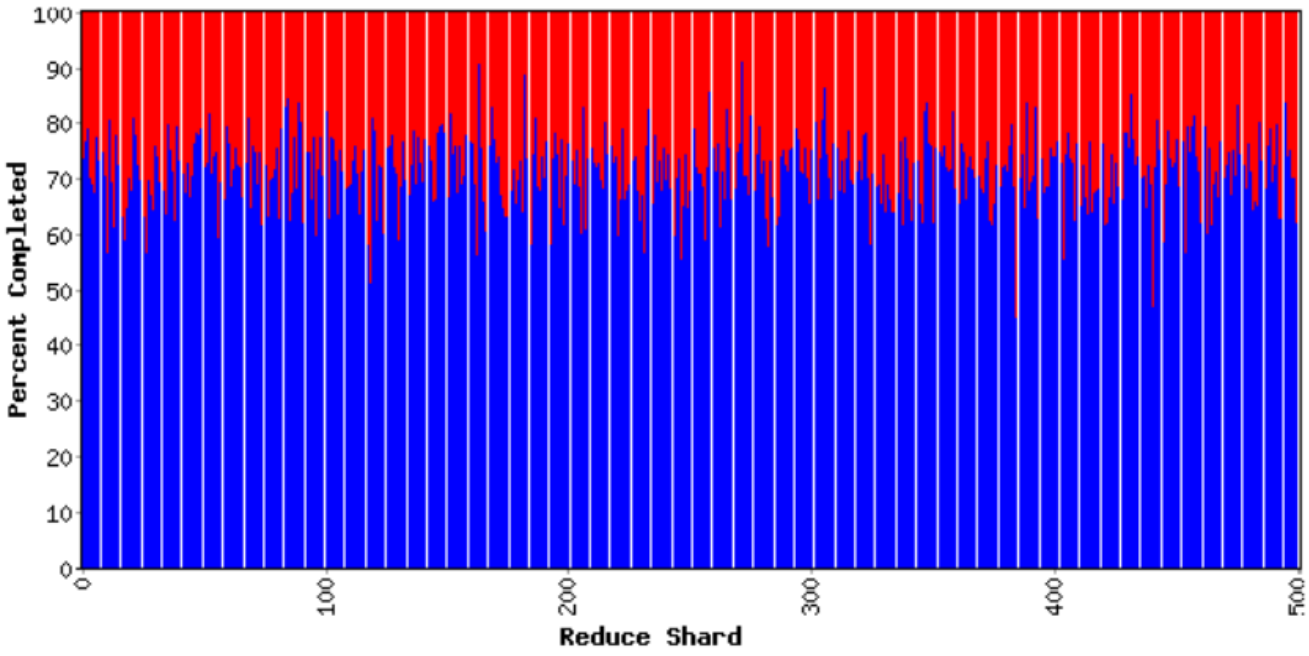
## OSDI 04 (Dean, Ghemawat)

### 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 |

# MapReduce

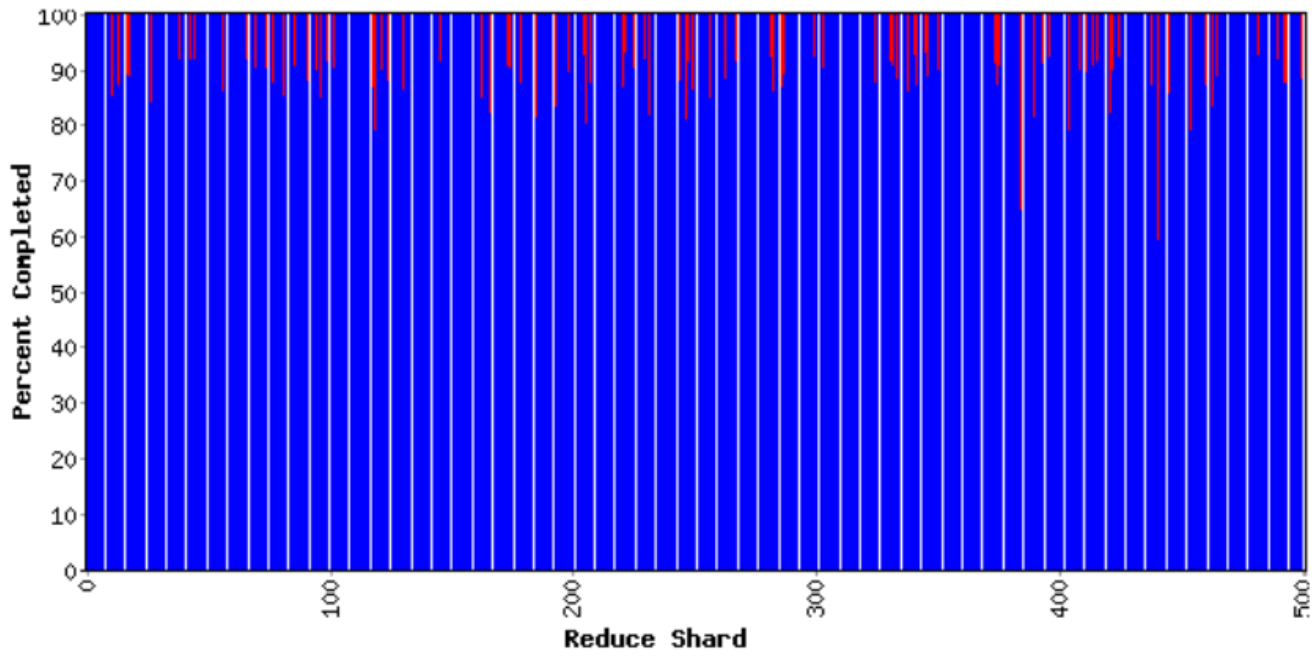
## OSDI 04 (Dean, Ghemawat)

### 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 | 1( |
| docs-indexed | 0 | |
| dups-in-index-merge | 0 | |
| mr-merge-calls | 35083350 | |
| mr-merge-outputs | 35083350 | |

# MapReduce

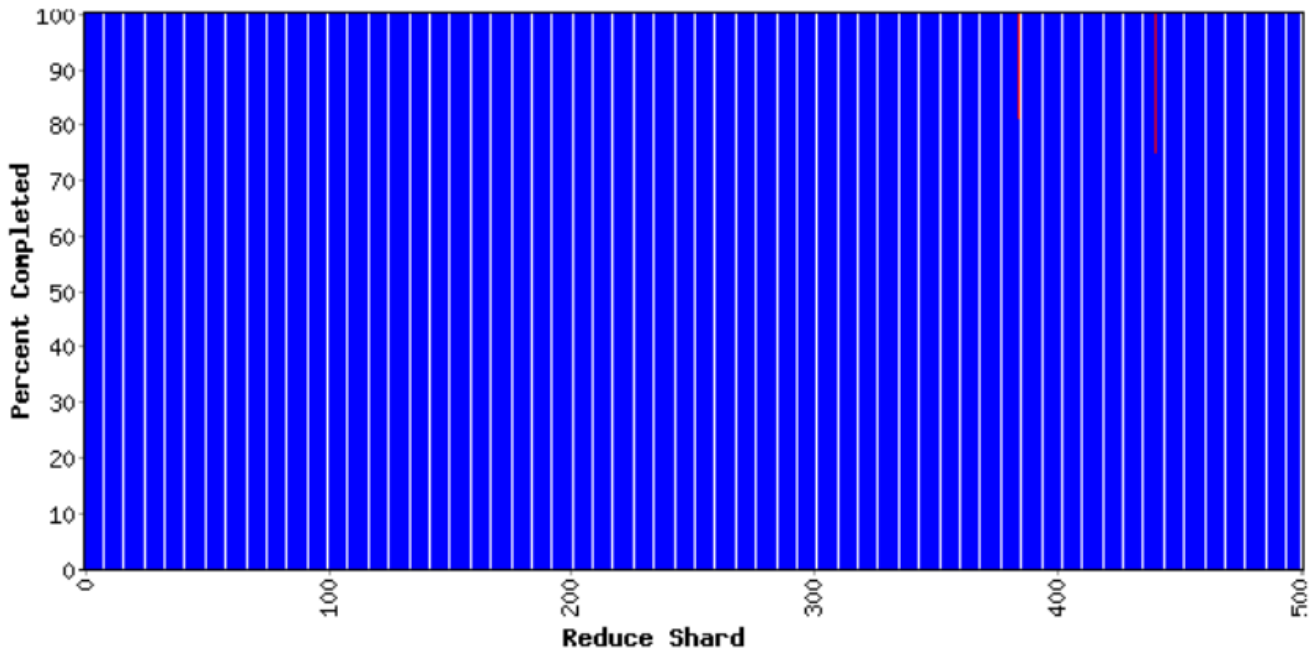
## OSDI 04 (Dean, Ghemawat)

### 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 |

# MapReduce

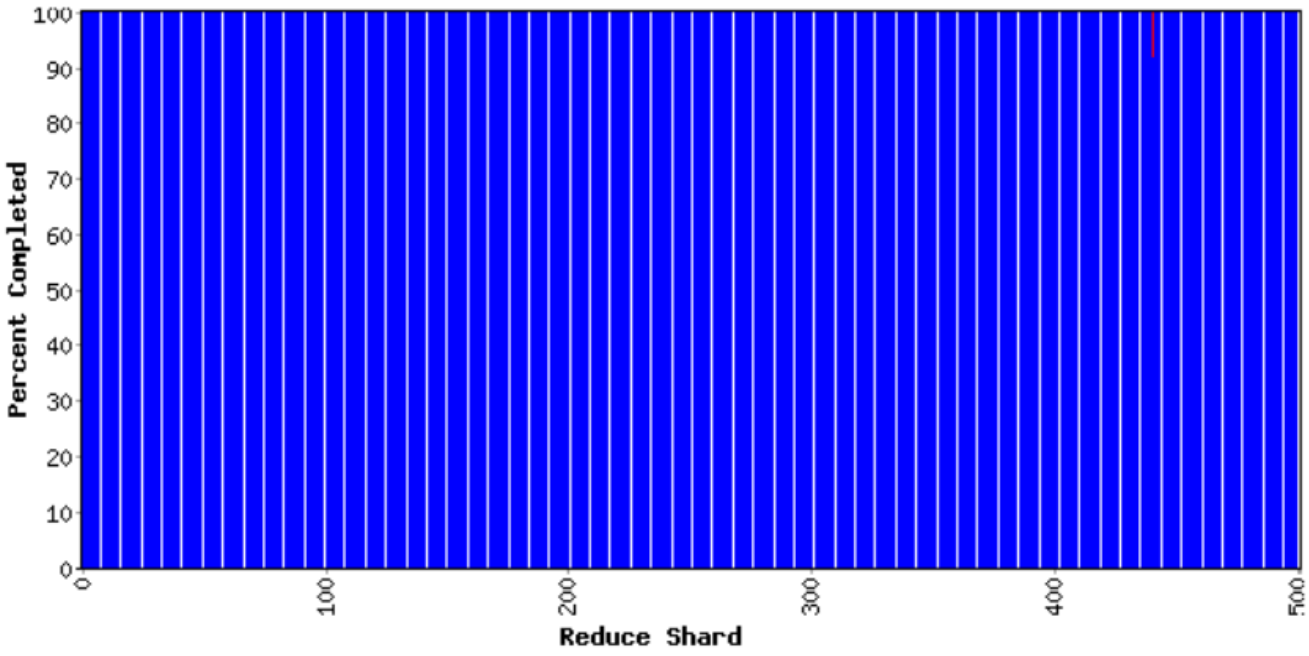
## OSDI 04 (Dean, Ghemawat)

### 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 | |

# Last Reducer

Typically Map phase linear on blocks. Reducers more variable.

No answer until the last one is done!
Some machines get slow/crash!

Solution: Automatically run back-up copies. Take first to complete.

# Last Reducer

Typically Map phase linear on blocks. Reducers more variable.

No answer until the last one is done!
Some machines get slow/crash!

Solution: Automatically run back-up copies. Take first to complete.

Scheduled by *Master Node*
Organizes computation, but does not process data.
If this fails, all goes down.

# Example 1: Word Count

Given text corpus $\langle$ doc id, list of words $\rangle$, count how many of each word exists.

**Map:**

# Example 1: Word Count

Given text corpus $\langle$ doc id, list of words $\rangle$, count how many of each word exists.

**Map:**
For each word $w \rightarrow \langle w, 1 \rangle$

**Reduce:**

# Example 1: Word Count

Given text corpus $\langle$ doc id, list of words $\rangle$, count how many of each word exists.

**Map:**
For each word $w \rightarrow \langle w, 1 \rangle$

**Reduce:**
$\{\langle w, c_1 \rangle, \langle w, c_2 \rangle, \langle w, c_3 \rangle, \ldots\} \rightarrow$

# Example 1: Word Count

Given text corpus $\langle$ doc id, list of words $\rangle$, count how many of each word exists.

**Map:**
For each word $w \rightarrow \langle w, 1 \rangle$

**Reduce:**
$\{\langle w, c_1 \rangle, \langle w, c_2 \rangle, \langle w, c_3 \rangle, \ldots\} \rightarrow \langle w, \sum_i c_i \rangle$

# Example 1: Word Count

Given text corpus $\langle$ doc id, list of words $\rangle$, count how many of each word exists.

**Map:**
For each word $w \rightarrow \langle w, 1 \rangle$

**Reduce:**
$\{\langle w, c_1 \rangle, \langle w, c_2 \rangle, \langle w, c_3 \rangle, \ldots\} \rightarrow \langle w, \sum_i c_i \rangle$

$w =$ "the" is 7% of all words!

# Example 1: Word Count

Given text corpus $\langle$ doc id, list of words $\rangle$, count how many of each word exists.

**Map:**
For each word $w \rightarrow \langle w, 1 \rangle$

**Reduce:**
$\{\langle w, c_1 \rangle, \langle w, c_2 \rangle, \langle w, c_3 \rangle, \ldots\} \rightarrow \langle w, \sum_i c_i \rangle$

$w =$ "the" is 7% of all words!
**Combine:** (before Map goes to Shuffle phase)
$\{\langle w, c_1 \rangle, \langle w, c_2 \rangle, \langle w, c_3 \rangle, \ldots\} \rightarrow \langle w, \sum_i c_i \rangle$

# Example 1: Word Count - Actual Code

```
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.mapreduce.*;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.TextInputFormat;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
```

# Example 1: Word Count - Actual Code

```
public class WordCount {

 public static class Map extends 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, Context context) throws IOException,
InterruptedException {
        String line = value.toString();
        StringTokenizer tokenizer = new StringTokenizer(line);
        while (tokenizer.hasMoreTokens()) {
            word.set(tokenizer.nextToken());
            context.write(word, one);
        }
    }
 }
```

# Example 1: Word Count - Actual Code

```
public static class Reduce extends Reducer<Text, IntWritable, Text, IntWritable> {

   public void reduce(Text key, Iterator<IntWritable> values, Context context)
     throws IOException, InterruptedException {
       int sum = 0;
       while (values.hasNext()) {
           sum += values.next().get();
       }
       context.write(key, new IntWritable(sum));
   }
}
```

# Example 1: Word Count - Actual Code

```
public static void main(String[] args) throws Exception {
    Configuration conf = new Configuration();

        Job job = new Job(conf, "wordcount");

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

    job.setMapperClass(Map.class);
    job.setReducerClass(Reduce.class);

    job.setInputFormatClass(TextInputFormat.class);
    job.setOutputFormatClass(TextOutputFormat.class);

    FileInputFormat.addInputPath(job, new Path(args[0]));
    FileOutputFormat.setOutputPath(job, new Path(args[1]));

    job.waitForCompletion(true);
}

}
```

job.setCombinerClass(Reduce.class);

# Example 2: Inverted Index

Given all of Wikipedia (all webpages), for each word, list all pages it is on.

**Map:**

# Example 2: Inverted Index

Given all of Wikipedia (all webpages), for each word, list all pages it is on.

**Map:**
For page $p$, each word $w \rightarrow \langle w, p \rangle$

**Reduce:**

# Example 2: Inverted Index

Given all of Wikipedia (all webpages), for each word, list all pages it is on.

**Map:**
For page $p$, each word $w \rightarrow \langle w, p \rangle$

**Reduce:**
$\{\langle w, p_1 \rangle, \langle w, p_2 \rangle, \langle w, p_3 \rangle, \ldots\} \rightarrow$

# Example 2: Inverted Index

Given all of Wikipedia (all webpages), for each word, list all pages it is on.

**Map:**
For page $p$, each word $w \rightarrow \langle w, p \rangle$

**Reduce:**
$\{\langle w, p_1 \rangle, \langle w, p_2 \rangle, \langle w, p_3 \rangle, \ldots\} \rightarrow \langle w, \cup_i p_i \rangle$

# Example 2: Inverted Index

Given all of Wikipedia (all webpages), for each word, list all pages it is on.

**Map:**
For page $p$, each word $w \rightarrow \langle w, p \rangle$

**Combine:**
$\{\langle w, p_1 \rangle, \langle w, p_2 \rangle, \langle w, p_3 \rangle, \ldots\} \rightarrow \langle w, \cup_i p_i \rangle$

**Reduce:**
$\{\langle w, p_1 \rangle, \langle w, p_2 \rangle, \langle w, p_3 \rangle, \ldots\} \rightarrow \langle w, \cup_i p_i \rangle$

# Hadoop

Open source version of MapReduce (and related, e.g. HDFS)

- Began 2005 (Cutting + Cafarella) supported by Yahoo!
- Stable enough for large scale around 2008
- Source code released 2009

Java (MapReduce in C++)

Led to *widespread* adoption in industry and academia!

# Rounds

Many algorithms are iterative, especially machine learning / data mining:

- Lloyd's algorithm for $k$-means
- gradient descent
- singular value decomposition

May require $\log_2 n$ rounds.

# Rounds

Many algorithms are iterative, especially machine learning / data mining:

- Lloyd's algorithm for $k$-means
- gradient descent
- singular value decomposition

May require $\log_2 n$ rounds. $\log_2(n = 1 \text{ billion}) \approx 30$

# Rounds

Many algorithms are iterative, especially machine learning / data mining:

- Lloyd's algorithm for $k$-means
- gradient descent
- singular value decomposition

May require $\log_2 n$ rounds. $\log_2(n = 1 \text{ billion}) \approx 30$

MapReduce puts rounds at a premium.
Hadoop can have several minute delay between rounds.
(Each rounds writes to HDFS for resiliency; same in MapReduce)

# Rounds

Many algorithms are iterative, especially machine learning / data mining:

- Lloyd's algorithm for $k$-means
- gradient descent
- singular value decomposition

May require $\log_2 n$ rounds. $\log_2(n = 1 \text{ billion}) \approx 30$

MapReduce puts rounds at a premium.
Hadoop can have several minute delay between rounds.
(Each rounds writes to HDFS for resiliency; same in MapReduce)

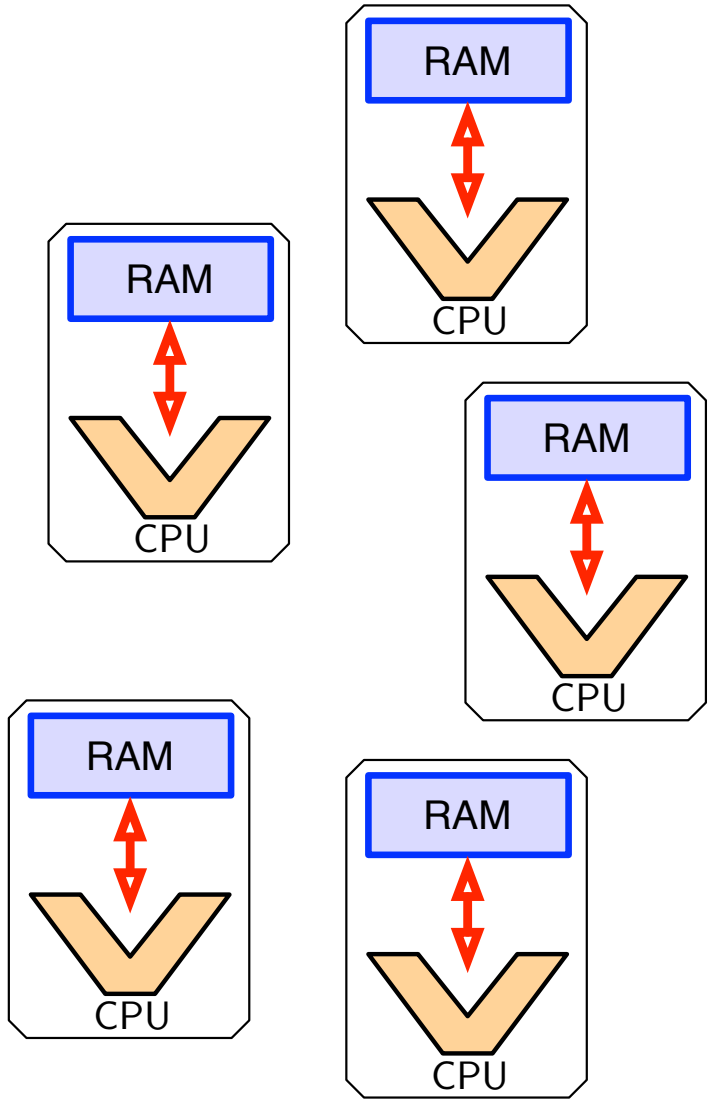
MRC Model (Karloff, Suri, Vassilvitskii; SODA 2010).
Stresses Rounds.

# Bulk Synchronous Parallel

Les Valiant [1989] BSP
Creates "barriers" in parallel algorithm.

1. Each processor computes on data
2. Processors send/receive data
3. Barrier : All processors wait for communication to end globally

Allows for easy synchronization. Easier to analyze since handles many messy synchronization details if this is emulated.

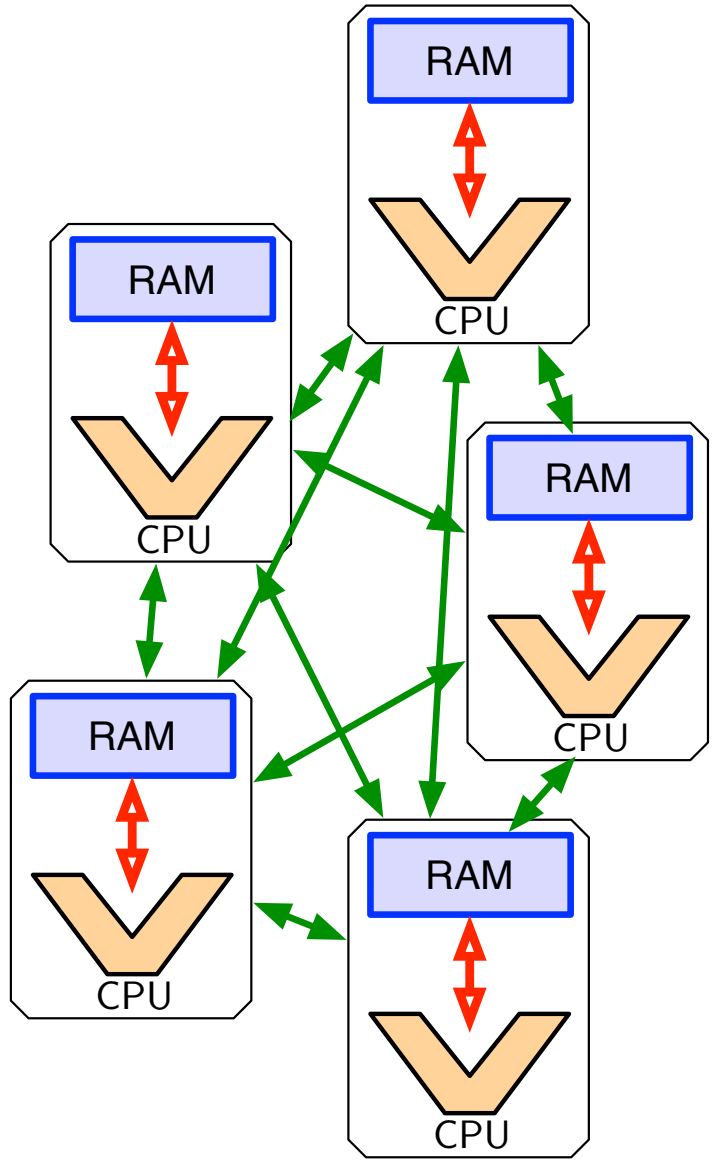


# Bulk Synchronous Parallel

Les Valiant [1989] BSP
Creates "barriers" in parallel algorithm.

1. Each processor computes on data
2. Processors send/receive data
3. Barrier : All processors wait for communication to end globally

Allows for easy synchronization. Easier to analyze since handles many messy synchronization details if this is emulated.

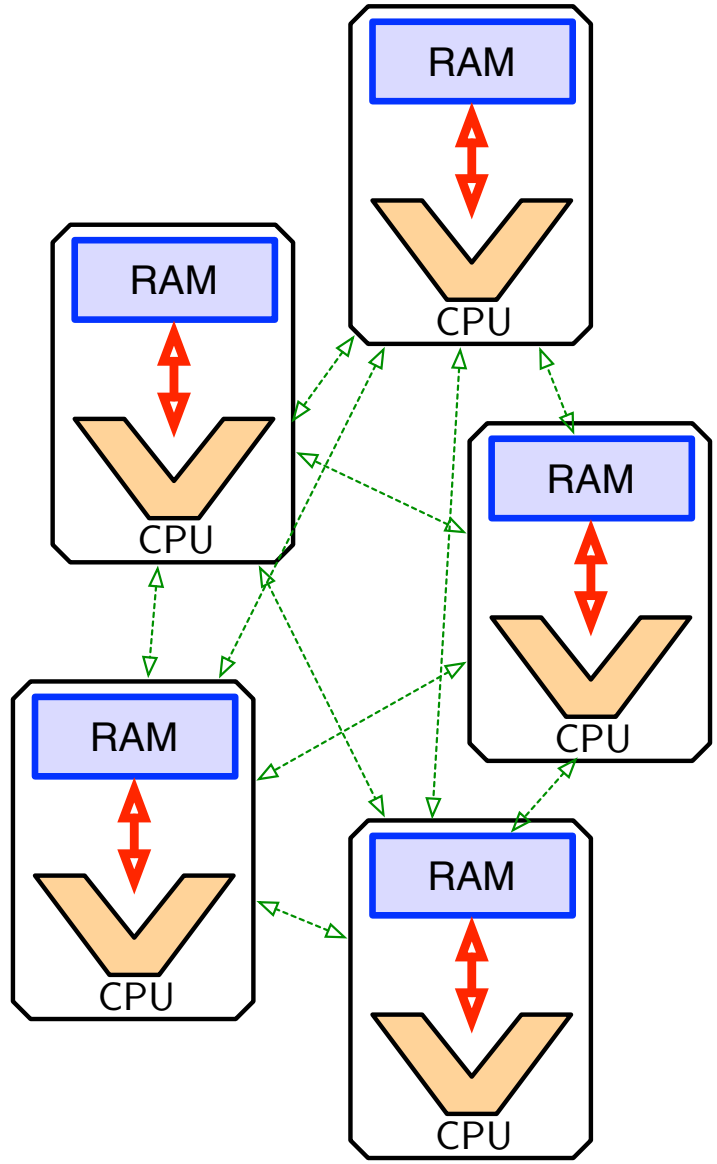


# Bulk Synchronous Parallel

Les Valiant [1989] BSP
Creates "barriers" in parallel algorithm.

1. Each processor computes on data
2. Processors send/receive data
3. Barrier : All processors wait for communication to end globally

Allows for easy synchronization. Easier to analyze since handles many messy synchronization details if this is emulated.




Reduction from MR (Goodrich, Sitchinava, Zhang; ISAAC 2011)

# Replication Rate

Consider a *join* of two sets of size $R, S$ of size $n = 10,000$.
List pair $(r, s) \in R \times S$ if $f(r, s) = 1$, for some function $f$.

**Option 1:** Create $n^2$ reducers.
Replication rate of $g = n$.

# Replication Rate

Consider a *join* of two sets of size $R, S$ of size $n = 10,000$.
List pair $(r, s) \in R \times S$ if $f(r, s) = 1$, for some function $f$.

**Option 1:** Create $n^2$ reducers.
Replication rate of $g = n$.

**Option 2:** Create 1 reducers.
Reducer of size $2n$, has $n^2 = 100 million$ operation.
Replication rate $g = 1$. No parallelism.

# Replication Rate

Consider a *join* of two sets of size $R, S$ of size $n = 10,000$.
List pair $(r, s) \in R \times S$ if $f(r, s) = 1$, for some function $f$.

**Option 1:** Create $n^2$ reducers.
Replication rate of $g = n$.

**Option 2:** Create 1 reducers.
Reducer of size $2n$, has $n^2 = 100 million$ operation.
Replication rate $g = 1$. No parallelism.

**Option 3:** Create $g^2$ reducers, each with 2 groups of size $n/g$.
Reducer size $2n/g$, $(n/g)^2$ operations ($g = 10$ only $1 million$).
Replication rate of $g$.

# Replication Rate

Consider a *join* of two sets of size $R, S$ of size $n = 10,000$.
List pair $(r, s) \in R \times S$ if $f(r, s) = 1$, for some function $f$.

**Option 1:** Create $n^2$ reducers.
Replication rate of $g = n$.

**Option 2:** Create 1 reducers.
Reducer of size $2n$, has $n^2 = 100 million$ operation.
Replication rate $g = 1$. No parallelism.

**Option 3:** Create $g^2$ reducers, each with 2 groups of size $n/g$.
Reducer size $2n/g$, $(n/g)^2$ operations ($g = 10$ only $1 million$).
Replication rate of $g$.

(Afrati, Das Sarma, Salihoglu, Ullman 2013),
(Beame, Koutris, Suciu 2013)

# Sawzall / Dremel / Tenzing

Google solution to *many to few*:

- Compute statistics on massive distributed data.
- Separates local computation from aggregation.
- Better with Iteration

| | Sawzall | Tenzing | Dremel |
|---|---|---|---|
| Latency | high | med | low |
| Scalability | high | high | med |
| SQL | none | high | med |
| Power | high | med | low |

# Berkeley Spark: Processing in memory

Zaharia, Chowdhury, Das, Ma, McCauley, Franklin, Shenker, Stoica (HotCloud 2010, NSDI 2012)

- Keeps relevant information in memory.
- Much faster on iterative algorithms (machine learning, SQL queries)
- Requires careful work to retain resiliency

Key idea: *RDDs: Resilient Distributed Data*. Can be stored in memory without replication, rebuilt from lineage if lost.