# Hadoop & MapReduce A new Paradigm for Distribution

Based on slides by
Hendra Setiawan, Owen O' Malley, and Jure Leskovec

Universität Zürich^UZH

Dynamic and Distributed Information Systems

# Using Commodity Clusters

## Problem

- How do you scale up applications?
  - Run jobs processing 100’s of terabytes of data
  - Takes 11 days to read on 1 computer
- Need lots of cheap computers
  - Fixes speed problem (15 minutes on 1000 computers), but…
  - Reliability problems
    - In large clusters, computers fail every day
    - Cluster size is not fixed
- Need common infrastructure
  - Must be efficient and reliable

## Problem in practice … ☹

- Google:
  - 20+ billion web-pages x 20 KB = 400+ TB
  - 1 Computer reads 30-35 MB/sec from disk
    - ~4 months to read the web
    - 1’000 HD to store the web
  - But both Computers and HDs fail…
    - If a HD or Computer fails every 1000 days, then
      - one HD is dead at any given point in time…
      - in 2006 Google had ~450’000 machines: 450 machines are down every day

# Challenges in Distributed Systems

- Transparency
  - Single view of the system
  - Hide numerous details
- Heterogeneity
  - Networks
  - Computers (HW)
  - Operating systems
  - Programming languages
  - Developers
- Failure Handling
  - Detecting
  - Masking
  - Tolerating
  - Recovery
  - Redundancy
- Openness
  - Extensibility
  - Publication of interfaces
- Scalability
  - Controlling the cost of resources
  - Controlling the performance
  - Preventing resources from running out
  - Avoiding performance bottlenecks
- Security
  - Secrecy
  - Authentication
  - Authorization
  - Non-repudiation
  - Mobile code
  - Denial of service

## So distributed Systems are complicated…

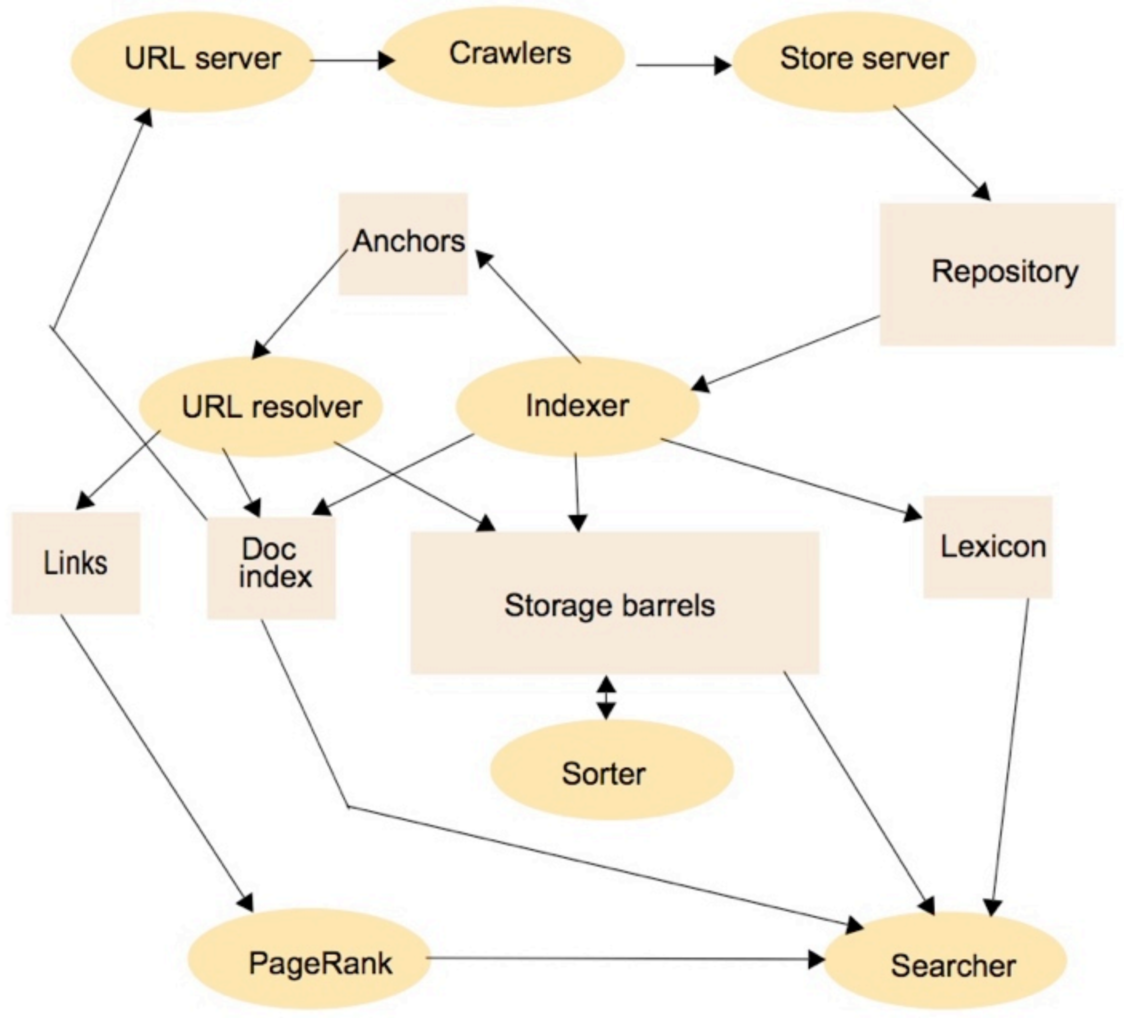
- … but many application require plug-in scalability

- http://www.facebook.com/video/video.php?v=10150555918930484

## Solution

- Use a commodity cluster and distribute data and tasks automagically….
- We need:
  - Really large data storage
  - A programming model & execution environment
  - A really large database
  - Distributed Coordination Services

## Outline architecture of the original Google search engine



## Agenda

- Challenges in Distributed Systems
- Using Commodity Clusters
  - **Really Large File System: HDFS**
  - Programming Abstractions for Seamless Distribution: Bulk Synchronous Parallel & MapReduce
  - Really Large Databases: HBase
  - Distributed Coordination Services: Zookeeper
- The Future:
  - Signal/Collect

## MapReduce

A Programing Abstraction for Synchronous Cluster Computing (Bulk Synchronous Parallel)

Universität Zürich UZH

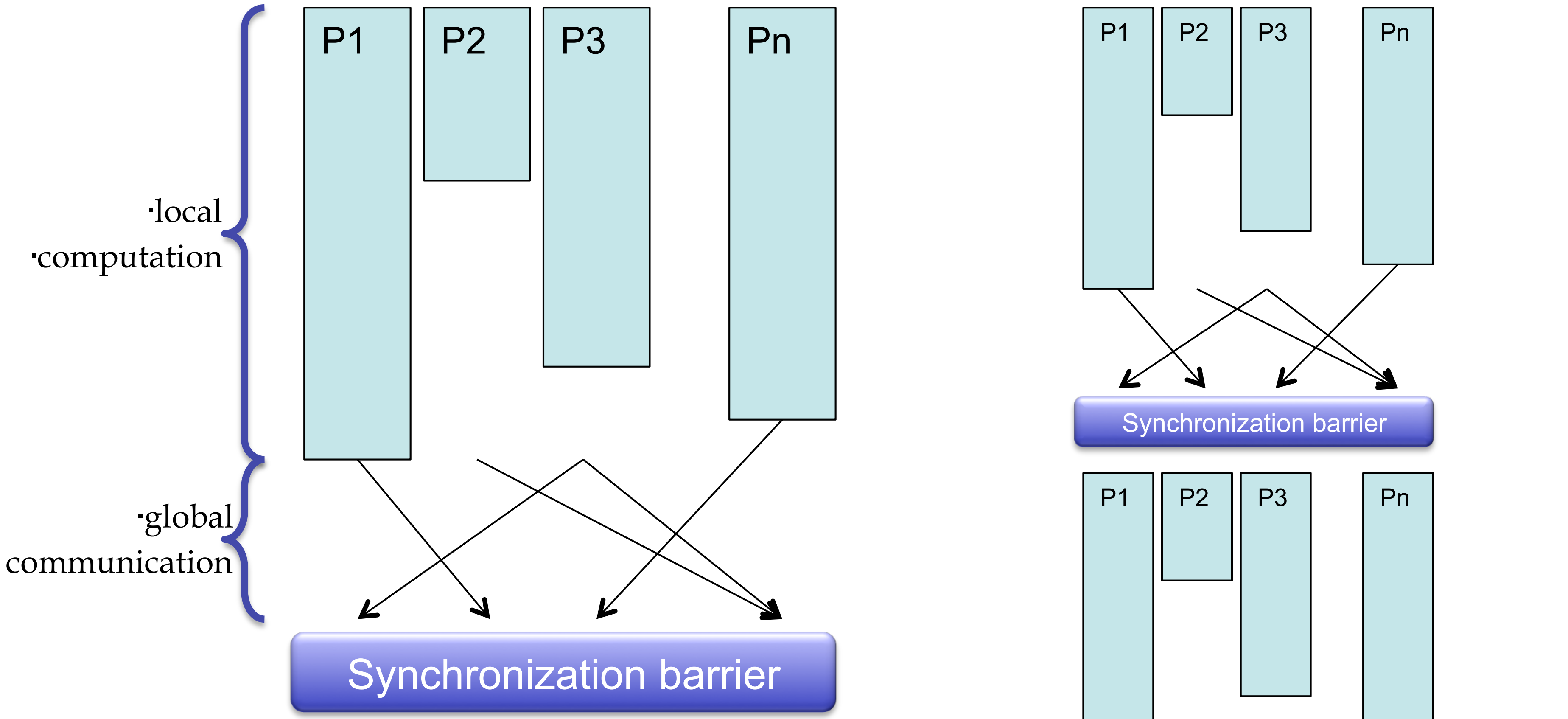
Dynamic and Distributed Information Systems

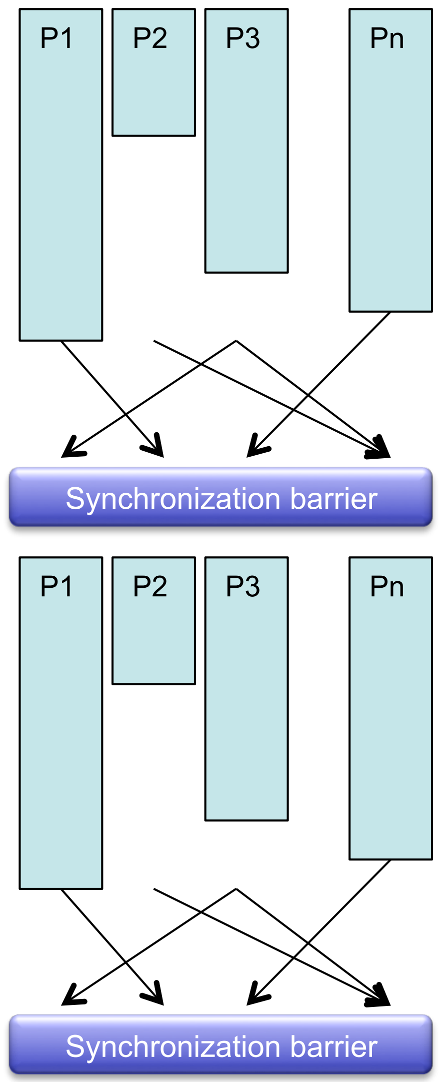
## Abstractions for Parallel Computing

- Computer Science is all about Abstractions
- What we are looking for is an abstraction that:

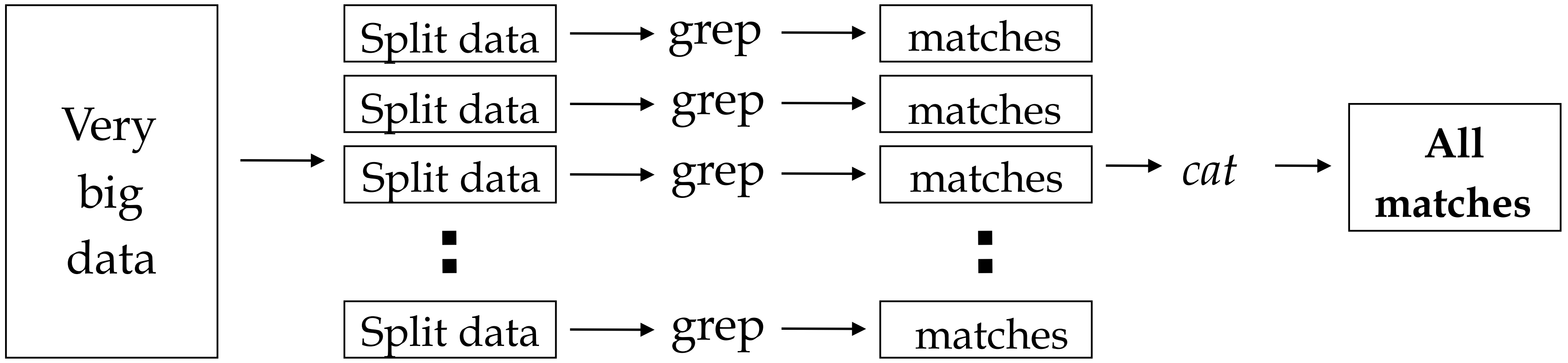
  "Automagically distributes (data and) tasks"
- Hence the abstraction should be:
  - Simple
  - Allow automatic distribution of tasks in a cluster
  - Is platform agnostic

# Bulk Synchronous Parallel (Valiant 1990)

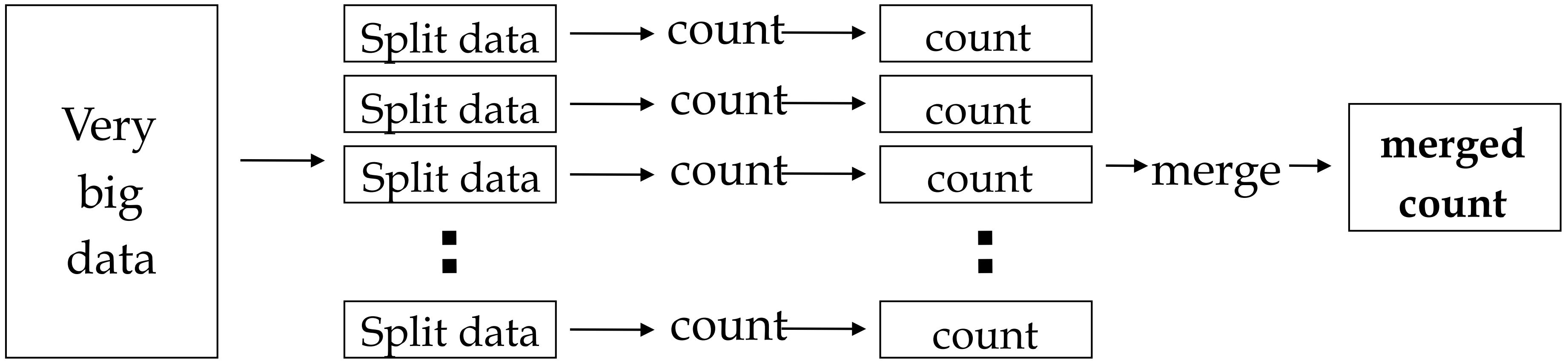




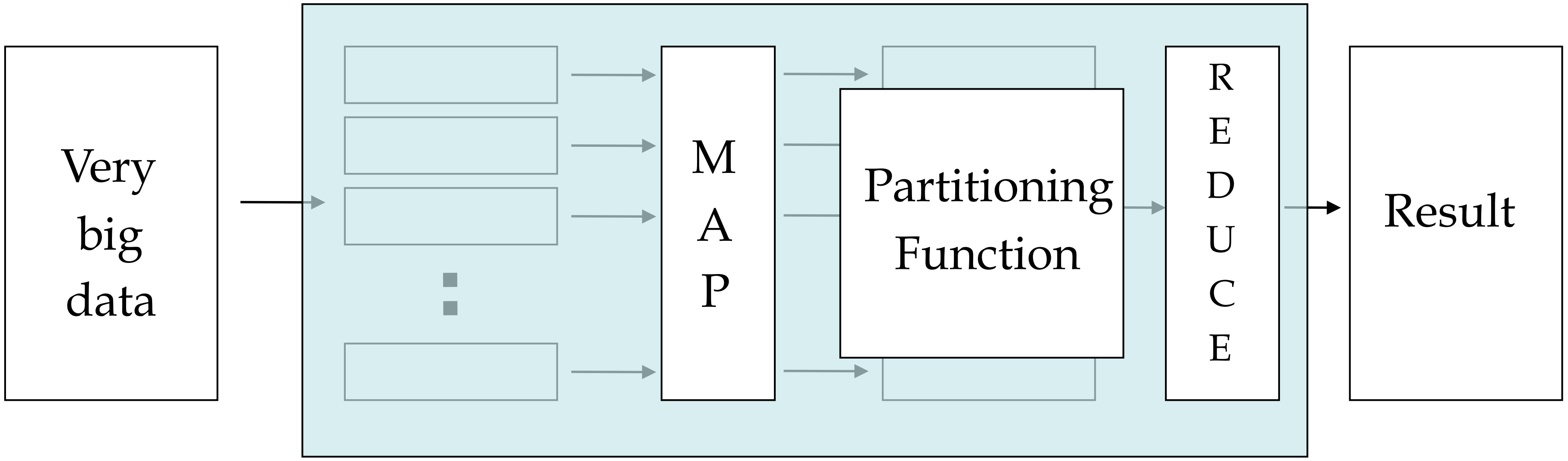
# Distributed Grep



## Distributed Word Count



## Map Reduce

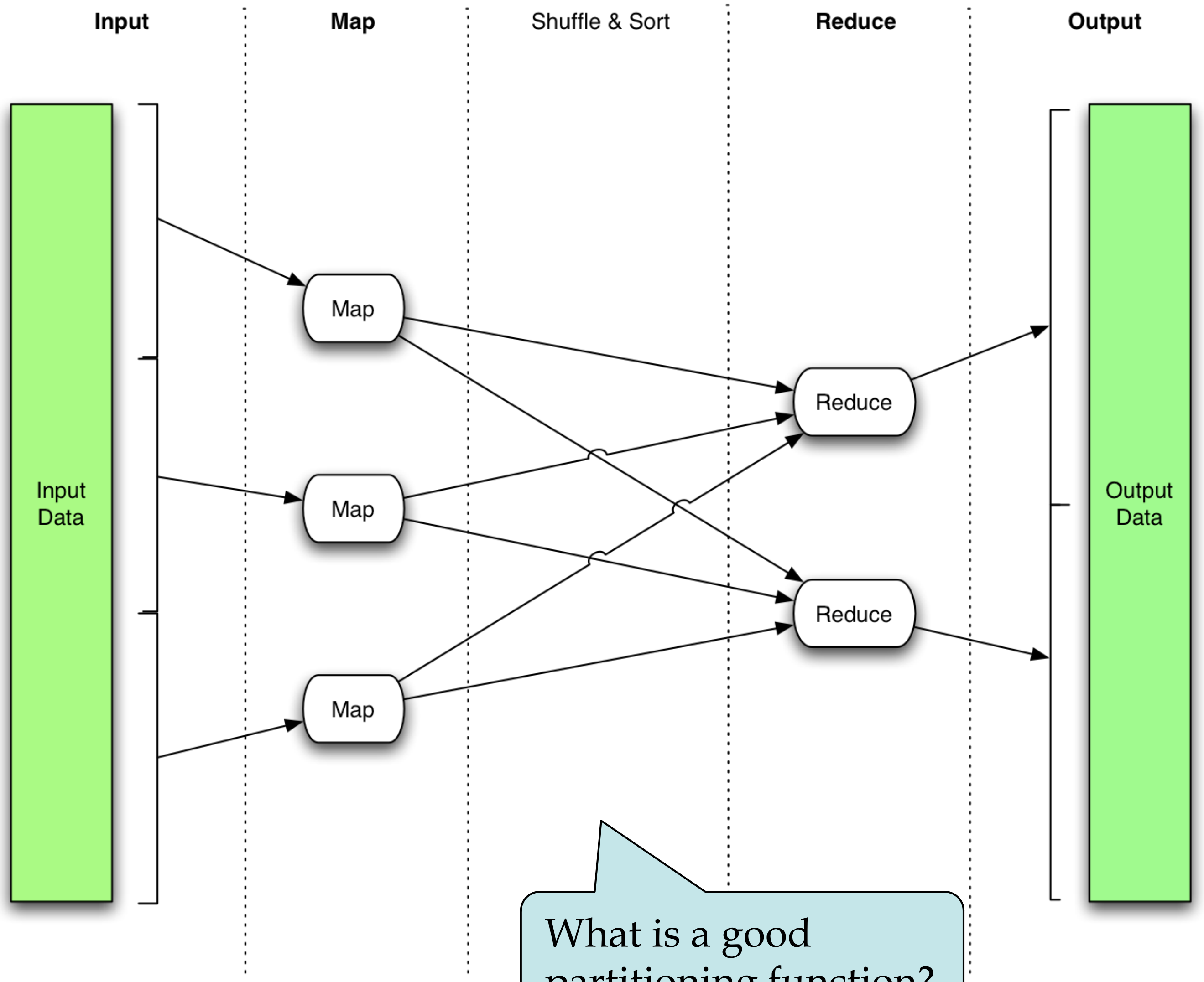


- Map:
  - Accepts *input* key/value pair
  - Emits *intermediate* key/value pair
- Reduce :
  - Accepts *intermediate* key/value* pair
  - Emits *output* key/value pair

## MapReduce in words

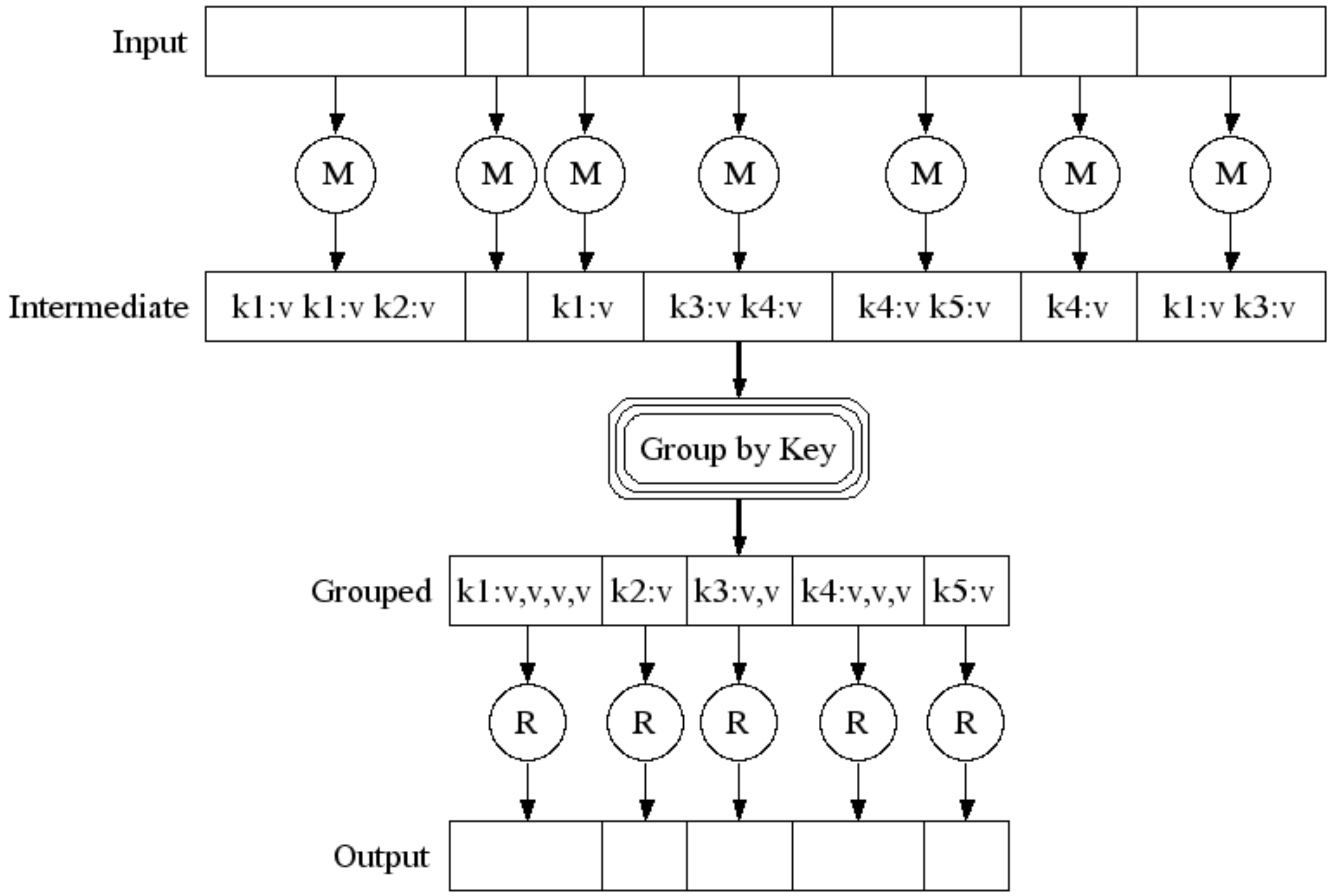
- MapReduce is a programming model for efficient distributed computing
- It works like a Unix pipeline:
  - cat input | grep | sort | uniq -c | cat > output
  - **Input** | **Map** | Shuffle & Sort | **Reduce** | **Output**
- Efficiency from
  - Streaming through data, reducing seeks
  - Pipelining
- A good fit for a lot of applications
  - Log processing
  - Web index building

# Map/Reduce Dataflow



What is a good partitioning function?

## Partitioning Function



## Partitioning Function (2)

- Default : `hash(key) mod R`
- Guarantee:
  - Relatively well-balanced partitions
  - Ordering guarantee within partition
- Distributed Sort
  - Map:

    ```
    emit(key,value)
    ```
  - Reduce (with R=1):

    ```
    emit(key,value)
    ```

## MapReduce

- Distributed Grep
  - Map:

```
if match(value,pattern) emit(value,1)
```

  - Reduce:

```
emit(key,sum(value*))
```

- Distributed Word Count
  - Map:

```
for all w in value do emit(w,1)
```

  - Reduce:

```
emit(key,sum(value*))
```

## MapReduce features

- Java and C++ APIs
  - In Java use Objects, while in C++ bytes
- Each task can process data sets larger than RAM
- Automatic re-execution on failure
  - In a large cluster, some nodes are always slow or flaky
  - Framework re-executes failed tasks
- Locality optimizations
  - Map-Reduce queries HDFS for locations of input data
  - Map tasks are scheduled close to the inputs when possible

## MapReduce Transparencies

Plus HDFS:

- Parallelization
- Fault-tolerance
- Locality optimization
- Load balancing

## Suitable for your task if

- Have a cluster
- Working with large dataset
- Working with independent data (or assumed)
- Can be cast into *map* and *reduce*

## MapReduce outside Google

- Hadoop (Java)
  - Emulates MapReduce and GFS
- The architecture of Hadoop MapReduce and DFS is master/slave

| | Master | Slave |
|---|---|---|
| MapReduce | jobtracker | tasktracker |
| DFS | namenode | datanode |

## Example Word Count (1)

- Map

```
public static class MapClass extends MapReduceBase
implements Mapper {
  private final static IntWritable one = new IntWritable(1);
  private Text word = new Text();

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

## Example Word Count (2)

- Reduce

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

## Example Word Count (3)

- Main

```
public static void main(String[] args) throws IOException {
  //checking goes here
  JobConf conf = new JobConf();

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

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

  conf.setInputPath(new Path(args[0]));
  conf.setOutputPath(new Path(args[1]));

  JobClient.runJob(conf);
}
```

## Practical MapReduce: Hadoop

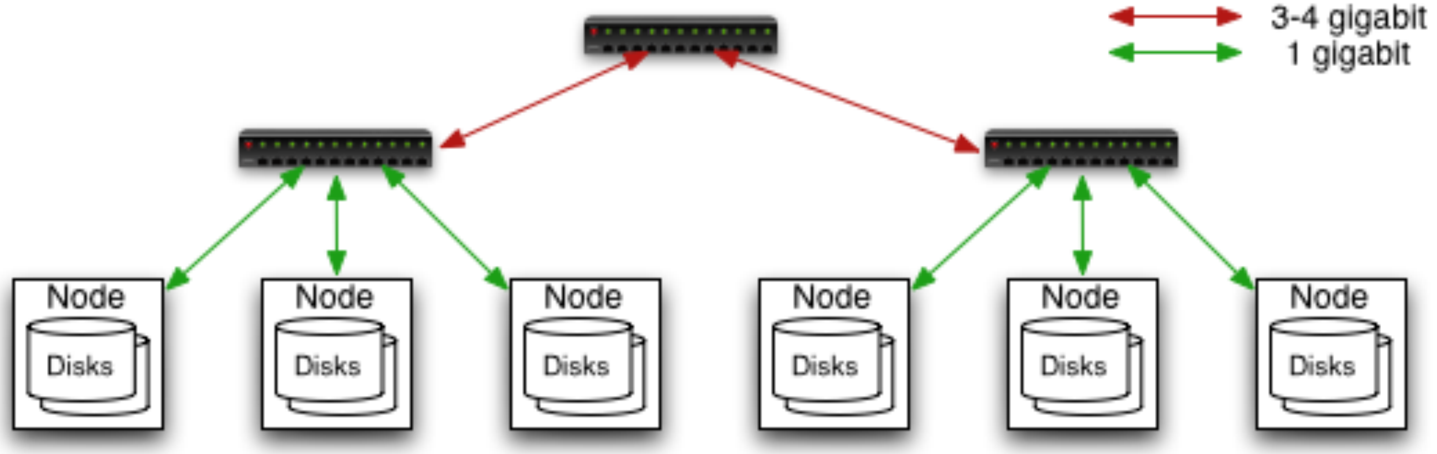
Universität Zürich UZH

Dynamic and Distributed Information Systems

## Solution

- Open Source Apache Project
- Hadoop Core includes:
  - Distributed File System - distributes data
  - Map/Reduce - distributes application
- Written in Java
- Runs on
  - Linux, Mac OS/X, Windows, and Solaris
  - Commodity hardware

# Commodity Hardware Cluster



- Typically in 2 level architecture
  - Nodes are commodity PCs
  - 40 nodes/rack
  - Uplink from rack is 8 gigabit
  - Rack-internal is 1 gigabit

## Distributed File System

- Single namespace for entire cluster
    - Managed by a single *namenode*.
    - Files are single-writer and append-only.
    - Optimized for streaming reads of large files.
- Files are broken in to large blocks.
    - Typically 128 MB
    - Replicated to several *datanodes*, for reliability
- Client talks to both namenode and datanodes
    - Data is not sent through the namenode.
    - Throughput of file system scales nearly linearly with the number of nodes.
- Access from Java, C, or command line.

## Block Placement

- Default is 3 replicas, but settable
- Blocks are placed (writes are pipelined):
  - On same node
  - On different rack
  - On the other rack
- Clients read from closest replica
- If the replication for a block drops below target, it is automatically re-replicated.

## Data Correctness

- Data is checked with CRC32
- File Creation
  - Client computes checksum per 512 byte
  - DataNode stores the checksum
- File access
  - Client retrieves the data and checksum from DataNode
  - If Validation fails, Client tries other replicas
- Periodic Validation

## How is Yahoo using Hadoop?

- They started with building better applications
  - Scale up web scale batch applications (search, ads, …)
  - Factor out common code from existing systems, so new applications will be easier to write
  - Manage the many clusters we have more easily
- The mission now includes research support
  - Build a **huge** data warehouse with many Yahoo! data sets
  - Couple it with a huge compute cluster and programming models to make using the data easy
  - Provide this as a service to our researchers
  - We are seeing great results!
    - Experiments can be run much more quickly in this environment

## Running Production WebMap

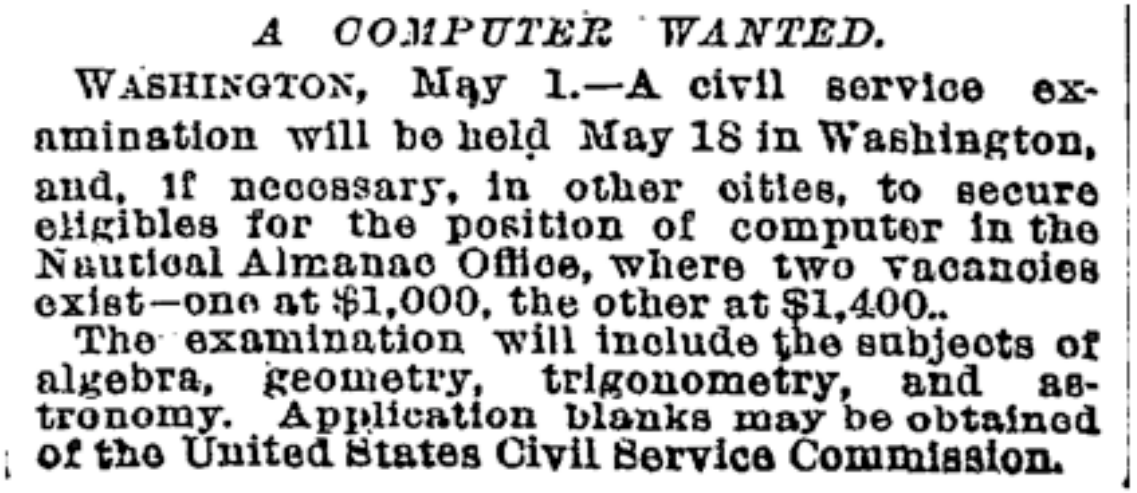
- Search needs a graph of the “known” web
    - Invert edges, compute link text, whole graph heuristics
- Periodic batch job using Map/Reduce
    - Uses a chain of ~100 map/reduce jobs
- Scale
    - 1 trillion edges in graph
    - Largest shuffle is 450 TB
    - Final output is 300 TB compressed
    - Runs on 10,000 cores
    - Raw disk used 5 PB
- Written mostly using Hadoop’s C++ interface

## Research Clusters

- The grid team runs the research clusters as a service to Yahoo researchers
- Mostly data mining/machine learning jobs
- Most research jobs are **not** Java:
  - 42% Streaming
    - Uses Unix text processing to define map and reduce
  - 28% Pig
    - Higher level dataflow scripting language
  - 28% Java
  - 2% C++

## NY Times

- Needed offline conversion of public domain articles from 1851-1922.
- Used Hadoop to convert scanned images to PDF
- Ran 100 Amazon EC2 instances for around 24 hours
- 4 TB of input
- 1.5 TB of output



*A COMPUTER WANTED.*

WASHINGTON, May 1.—A civil service examination will be held May 18 in Washington, and, if necessary, in other cities, to secure eligibles for the position of computer in the Nautical Almanac Office, where two vacancies exist—one at $1,000, the other at $1,400..

The examination will include the subjects of algebra, geometry, trigonometry, and astronomy. Application blanks may be obtained of the United States Civil Service Commission.

·Published 1892, copyright New York Times

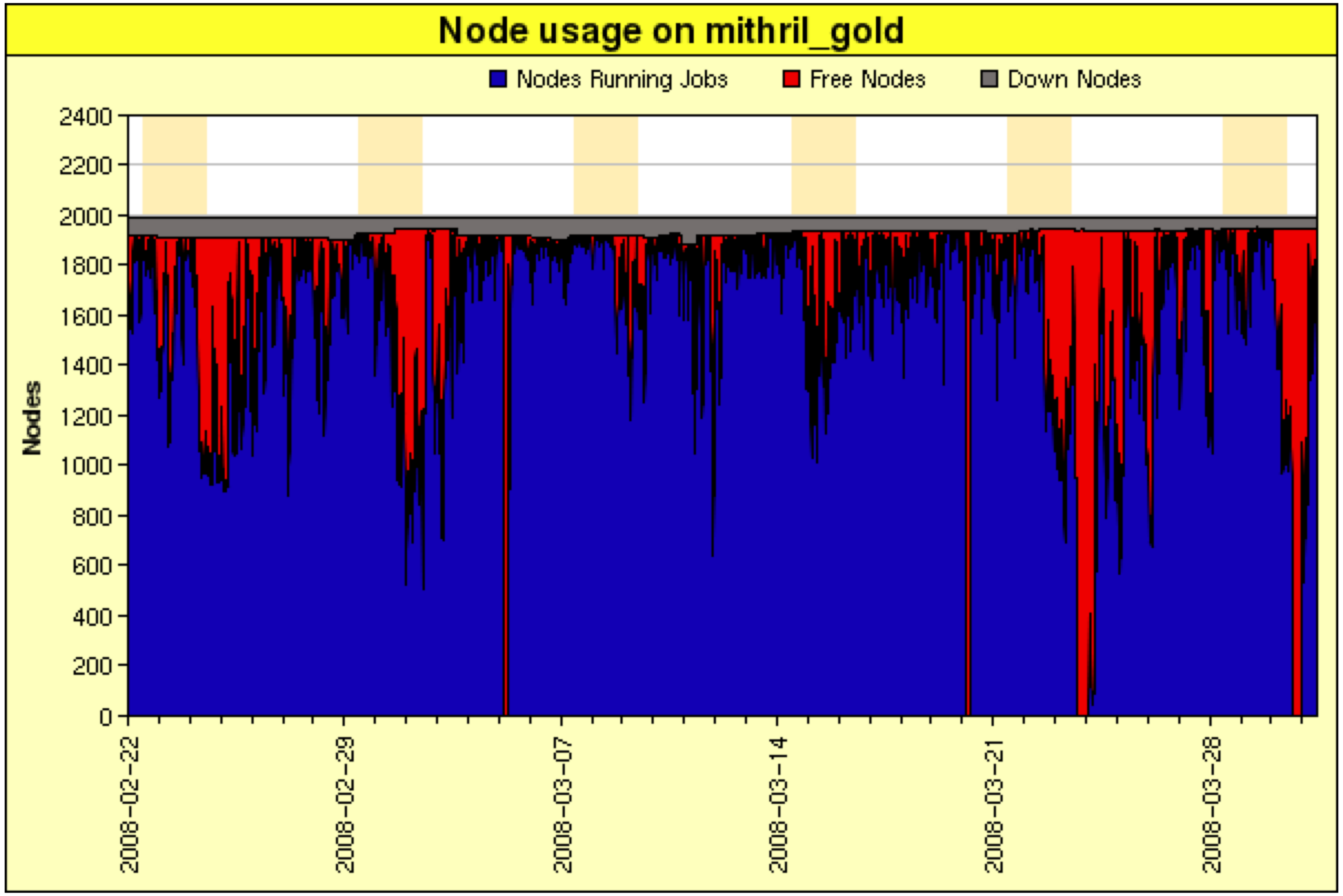
## Terabyte Sort Benchmark

- Started by Jim Gray at Microsoft in 1998
- Sorting 10 billion 100 byte records
- Hadoop won the general category in 209 seconds
  - 910 nodes
  - 2 quad-core Xeons @ 2.0Ghz / node
  - 4 SATA disks / node
  - 8 GB ram / node
  - 1 gb ethernet / node
  - 40 nodes / rack
  - 8 gb ethernet uplink / rack
- Previous records was 297 seconds
- Only hard parts were:
  - Getting a total order
  - Converting the data generator to map/reduce

## Hadoop clusters

- They have ~20,000 machines running Hadoop
- Largest clusters are currently 2000 nodes
- Several petabytes of user data (compressed, unreplicated)
- They run hundreds of thousands of jobs every month

## Research Cluster Usage



## Who Uses Hadoop?

- Amazon/A9
- AOL
- Facebook
- Fox interactive media
- Google / IBM
- New York Times
- PowerSet (now Microsoft)
- Quantcast
- Rackspace/Mailtrust
- Veoh
- Yahoo!
- More at http://wiki.apache.org/hadoop/PoweredBy

## References

- Original paper (http://labs.google.com/papers/mapreduce.html)
- On wikipedia (http://en.wikipedia.org/wiki/MapReduce)
- Hadoop – MapReduce in Java (http://lucene.apache.org/hadoop/)
- Starfish - MapReduce in Ruby (http://rufy.com/starfish/)

## Q&A - Hadoop

- For more information:
  - Website: http://hadoop.apache.org/core
  - Mailing lists:
    - core-dev@hadoop.apache
    - core-user@hadoop.apache
  - IRC: #hadoop on irc.freenode.org