On the Convergence of Cloud Computing and Desktop Grids

Presented by Derrick Kondo

Many Slides by

Jeff Barr, Amazon Inc.

and Jeff Dean, Sanjay Ghemawat, Google, Inc.
Outline

- **Cloud Computing**
  - Background
  - Architecture
  - Map-Reduce

- **Desktop Grids**
  - Background & contract with clouds
  - Architecture
  - Prediction
Motivation

- 70% of Web Development Effort is “Muck”:
  - Data Centers
  - Bandwidth / Power / Cooling
  - Operations
  - Staffing

- Scaling is Difficult and Expensive:
  - Large Up-Front Investment
  - Invest Ahead of Demand
  - Load is Unpredictable
Dream or Nightmare?

- Slashdot/Digg/TechCrunch Effect
- Rapid, unexpected customer demand/growth

Same true for scientific workloads

![Chart showing daily pageviews per million from April to June 2006 with a peak in May.](image)
Solution: Cloud Computing

- Scale capacity on demand
- Turn fixed costs into variable costs
- Always available
- Rock-solid reliability
- Simple APIs and conceptual models
- Cost-effective
- Reduced time to market
- Focus on product & core competencies
What is a cloud?

- Cloud computing is Internet-based ("cloud") development and use of computer technology ("computing"). -- Wikipedia

- A cloud is a distributed system where the user doesn't care exactly what resources are used to carry out an operation -- Prof. Douglas Thain

- "A Cloud is a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resources based on service-level agreements established through negotiation between the service provider and consumers.” -- Prof Raj Buyya
Cloud Providers

- Large-scale centralized systems
  - Low reliability, low-cost commodity components
- Google
  - 100,000 systems in 15 data centers [2005]
    - Recent estimate: 500,000 systems in 30 data centers
    - 1,152 systems in 20x8x8 foot container

Figure 5: Sun Microsystems Black Box
Types of Clouds

- **Platform-as-a-service**
  - E.g. Amazon’s EC2

- **Applications-as-a-service**
  - E.g. Google App Engine (DataStore/GQL, MapReduce)
Google App Engine

- Run web applications (Python-based)
- API for data store, google accounts, URL fetching, image manip., email
- Web-based admin console
- Free with up to 500MB of storage and 5 million views
Infrastructure Services

Compute

Store

Message
Infrastructure Services

- Compute
- Store
- Message
- Simple Storage Service
- Simple Queue Service
- Elastic Compute Cloud
Amazon Simple Storage Service

- 1 B – 5 GB / object
- Fast, Reliable, Scalable
- Redundant, Dispersed
- 99.99% Availability

Goal
- Private or Public
- Per-object URLs & ACLs
- BitTorrent Support
Pricing in Europe

Storage

* $0.180 per GB – first 50 TB / month of storage used
* $0.170 per GB – next 50 TB / month of storage used
* $0.160 per GB – next 400 TB / month of storage used
* $0.150 per GB – storage used / month over 500 TB

Data Transfer

* $0.100 per GB – all data transfer in
* $0.170 per GB – first 10 TB / month data transfer out
* $0.130 per GB – next 40 TB / month data transfer out
* $0.110 per GB – next 100 TB / month data transfer out
* $0.100 per GB – data transfer out / month over 150 TB

Requests

* $0.012 per 1,000 PUT, COPY, POST, or LIST requests
* $0.012 per 10,000 GET and all other requests*
Amazon S3 Concepts

Objects:
- Opaque data to be stored (1 byte … 5 Gigabytes)
- Metadata (attribute-value, up to 4KB)
- Authentication and access controls

Buckets (like directories):
- Object container – any number of objects
- 100 buckets per account / buckets are “owned”

Keys:
- Unique object identifier within bucket
- Up to 1024 bytes long
- Flat object storage model

Functionality
- Simple put/get functionality
- Limited search functionality
- Objects are immutable, cannot be renamed

Standards-Based Interfaces:
- REST and SOAP
- URL-Addressability – every object has a URL

2-level namespace
Make your photos come alive.

- **Unlimited** photos
- No ads or spam
- **Gorgeous** galleries

Try It!  
Learn more
Amazon Elastic Compute Cloud

EC2
Amazon EC2

- Virtual environment for linux/windows applications
  - Create Amazon Machine Image (AMI) with app’s, lib’s, data, config settings,
  - Upload image to S3, then start/stop/monitor images
Amazon EC2 Features

- Elastic: can increase number of resources as needed
- Configurability: can configure hard resources (as instances) or software stack: OS, lib’s, app’s with root access
- Reliability: 99.99%
- For applications
  - Persistent storage (independant of life of instance)
  - Multiple locations: availability zones
  - Static IP addresses associated with account (not instance)
    - Can remap IP addresses to another instance or availability zone as needed
Amazon EC2 Concepts

- **Amazon Machine Image (AMI):**
  - Bootable root disk
  - Pre-defined or user-built
  - Catalog of user-built AMIs

- **Instance:**
  - Running copy of an AMI
  - Launch in less than 2 minutes
  - Start/stop programmatically

- **Network Security Model:**
  - Explicit access control
  - Security groups

- Inter-service bandwidth is free
Standard Instances

- **Small Instance (Default)** 1.7 GB of memory, 1 EC2 Compute Unit (1 virtual core with 1 EC2 Compute Unit), 160 GB of instance storage, 32-bit platform
- **Large Instance** 7.5 GB of memory, 4 EC2 Compute Units (2 virtual cores with 2 EC2 Compute Units each), 850 GB of instance storage, 64-bit platform
- **Extra Large Instance** 15 GB of memory, 8 EC2 Compute Units (4 virtual cores with 2 EC2 Compute Units each), 1690 GB of instance storage, 64-bit platform

- **EC2 Compute Unit (ECU)** – One EC2 Compute Unit (ECU) provides the equivalent CPU capacity of a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor.
Instances of this family have proportionally more CPU resources than memory (RAM) and are well suited for compute-intensive applications.

- **High-CPU Medium Instance**: 1.7 GB of memory, 5 EC2 Compute Units (2 virtual cores with 2.5 EC2 Compute Units each), 350 GB of instance storage, 32-bit platform

- **High-CPU Extra Large Instance**: 7 GB of memory, 20 EC2 Compute Units (8 virtual cores with 2.5 EC2 Compute Units each), 1690 GB of instance storage, 64-bit platform
Operating Systems and Software

- Operating Systems
  - Red Hat Enterprise Linux  Windows Server 2003  Oracle Enterprise Linux
  - OpenSolaris  openSUSE Linux  Ubuntu Linux
  - Fedora  Gentoo Linux  Debian

- Software
  - Databases
    - Oracle 11g, MySQL Enterprise, Microsoft SQL Server Standard 2005
  - Batch Processing
    - Hadoop, Condor
  - eb Hosting
    - Apache HTTP, IIS/Asp.Net
Pricing

Pay as you use

Standard Instances

Linux

- Small (Default) $0.10 per hour
- Large $0.40 per hour
- Extra Large $0.80 per hour

High CPU Instances

- Medium $0.20 per hour
- Extra Large $0.80 per hour

Internet Data Transfer

- Data transfer in: $0.10 per GB
- Data transfer out: $0.17 per GB
Amazon EC2 At Work

- Startups
  - Cruxy – Media transcodin
g  
  - GigaVox Media – Podcast Management

- Fortune 500 clients:
  - High-Impact, Short-Term Projects
  - Development Host

- Science / Research:
  - Hadoop / MapReduce
  - mpiBLAST

- Load-Management and Load Balancing Tools:
  - Pound
  - Weogeo
  - Rightscale
Can Clouds Work for Science?

- Applications don’t need durability, availability, and access performance all bundled together

CPU costs dominate for scientific workflow application called montage

<table>
<thead>
<tr>
<th>Characteristics</th>
<th>Resources and techniques to provide them</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-performance data access</td>
<td>Geographical data (or storage) replication to improve access locality, high-speed storage, fat networks</td>
</tr>
<tr>
<td>Durability</td>
<td>Data replication - possible at various levels: hardware (RAID), multiple locations, multiple media; erasure codes</td>
</tr>
<tr>
<td>Availability</td>
<td>Server/service replication, hot-swap technologies, multi hosting, techniques to increase availability for auxiliary services (e.g., authentication, access control)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Application class</th>
<th>Durability</th>
<th>Availability</th>
<th>High access speed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cache</td>
<td>No</td>
<td>Depends</td>
<td>Yes</td>
</tr>
<tr>
<td>Long-term archival</td>
<td>Yes</td>
<td>No</td>
<td>No</td>
</tr>
<tr>
<td>Online production</td>
<td>No</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Batch production</td>
<td>No</td>
<td>No</td>
<td>Yes</td>
</tr>
</tbody>
</table>
MapReduce: Simplified Data Processing on

These are slides from Dan Weld’s class at U. Washington
(who in turn made his slides based on those by Jeff Dean, Sanjay Ghemawat, Google, Inc.)
An abstraction is a simple interface that allows you to scale up well-structured problems to run on hundreds or thousands of computers at once.

-- Douglas Thain
Large-scale Management Issues

- How to parallelize
- Data distribution
- Scheduling
- Load balancing
- Failure management
- Deployment
MapReduce

- MapReduce provides
  - Automatic parallelization & distribution
  - Fault tolerance
  - I/O scheduling
  - Monitoring & status updates
Map/Reduce

- Map/Reduce
  - Programming model from Lisp
  - (and other functional languages)
    - state what you want to do not how to get it
- Many problems can be phrased this way
- Easy to distribute across nodes
- Nice retry/failure semantics
Map in Lisp (Scheme)
Map in Lisp (Scheme)

- \((\text{map } f \ \text{list } [\text{list}_2 \ \text{list}_3 \ \ldots])\)
Map in Lisp (Scheme)

- `(map f list [list₂ list₃ ...])`
Map in Lisp (Scheme)

- `(map f list [list₂ list₃ ...])`
Map in Lisp (Scheme)

- `(map f list [list_2 list_3 ...])`
- `(map square '(1 2 3 4))`
Map in Lisp (Scheme)

- `(map f list [list_2 list_3 ...])`
- `(map square '(1 2 3 4))`

Unary operator

Binary operator
Map in Lisp (Scheme)

- `(map f list [list₂ list₃ ...])`
- `(map square '(1 2 3 4))`
Map in Lisp (Scheme)

- `(map f list [list₂ list₃ ...])`
- `(map square '(1 2 3 4))`
Map in Lisp (Scheme)

- `(map f list [list₂, list₃, ...])`
- `(map square '(1 2 3 4))`

Unary operator

Binary operator
Map in Lisp (Scheme)

- `(map f list [list₂ list₃ ...])`
- `(map square '(1 2 3 4))`
  - `(1 4 9 16)`

Unary operator

Binary operator
Map in Lisp (Scheme)

- `(map f list [list₂ list₃ ...])`
- `(map square '(1 2 3 4))`  
  - `(1 4 9 16)`
Map in Lisp (Scheme)

- `(map f list [list₂ list₃ ...])`
- `(map square '(1 2 3 4))`  
  - `(1 4 9 16)`
- `(reduce + '(1 4 9 16))`
Map in Lisp (Scheme)

- \((\text{map } f \text{ list } [\text{list}_2 \text{ list}_3 \ldots])\)

- \((\text{map square } '(1 2 3 4))\)
  - \((1 4 9 16)\)

- \((\text{reduce } + \ ('(1 4 9 16))\)
  - \((+ 16 (+ 9 (+ 4 1)))\)
Map in Lisp (Scheme)

- \((\text{map } f \ \text{list} \ [\text{list}_2\ \text{list}_3\ ...])\)

- \((\text{map} \ \text{square} \ '(1\ 2\ 3\ 4))\)
  - \((1\ 4\ 9\ 16)\)

- \((\text{reduce} + \ '(1\ 4\ 9\ 16))\)
  - \((+\ 16\ (+\ 9\ (+\ 4\ 1)))\)
  - \(30\)
Map/Reduce ala Google
Map/Reduce ala Google

- **map(key, val)** is run on each item in set
  - emits new key, val pairs

- **reduce(key, vals)** is run for each unique key emitted by map()
  - emits final output
count words in docs
count words in docs

- Input consists of (url, contents) pairs
count words in docs

- Input consists of (url, contents) pairs

- map(key=url, val=contents):
  - For each word w in contents, emit (w, “1”)
count words in docs

- Input consists of (url, contents) pairs

- map(key=url, val=contents):
  - For each word $w$ in contents, emit ($w$, “1”)

- reduce(key=word, values=uniq_counts):
count words in docs

- **Input consists of** *(url, contents)* **pairs**

- **map***(key=url, val=contents)*:
  - For each word *w* in contents, emit *(w, "1")*

- **reduce***(key=word, values=uniq_counts)*:
  - Sum all "1"s in values list
count words in docs

- Input consists of (url, contents) pairs

- map(key=url, val=contents):
  - For each word w in contents, emit (w, “1”)

- reduce(key=word, values=uniq_counts):
  - Sum all “1”s in values list
  - Emit result “(word, sum)”
Count,

map(key=url, val=contents):
   For each word w in contents, emit (w, "1")
reduce(key=word, values=uniq_counts):
   Sum all "1"s in values list

see bob throw
see spot run
Count,

map(key=url, val=contents):
    For each word w in contents, emit (w, "1")

reduce(key=word, values=uniq_counts):
    Sum all "1"s in values list

see bob throw
see spot run

see 1
bob 1
run 1
see 1
spot 1
throw 1
Count,

map(key=url, val=contents):
For each word \( w \) in contents, emit \((w, "1")\)

reduce(key=word, values=uniq_counts):
Sum all "1"s in values list

```
see bob throw
see spot run

see  1  bob  1
bob  1  run  1
run  1  see  2
see  1  spot  1
spot 1  throw 1
throw 1
```
Grep
Grep

- Input consists of (url+offset, single line)
- map(key=url+offset, val=line):
Grep

- Input consists of (url+offset, single line)
- map(key=url+offset, val=line):
  - If contents matches regexp, emit (line, "1")
Grep

- Input consists of (url+offset, single line)
- map(key=url+offset, val=line):
  - If contents matches regexp, emit (line, “1”)

- reduce(key=line, values=uniq_counts):
Grep

- Input consists of (url+offset, single line)
- map(key=url+offset, val=line):
  - If contents matches regexp, emit (line, “1“)

- reduce(key=line, values=uniq_counts):
  - Don’t do anything; just emit line
Model is Widely Applicable

Example uses:
- distributed grep
- distributed sort
- web link-graph reversal
- term-vector / host
- web access log stats
- inverted index construction
- document clustering
- machine learning
- statistical machine translation
Implementation Overview

Typical cluster:

• 100s/1000s of 2-CPU x86 machines, 2-4 GB of memory
• Limited bisection bandwidth
• Storage is on local IDE disks
• GFS: distributed file system manages data (SOSP'03)
• Job scheduling system: jobs made up of tasks, scheduler assigns tasks to machines

Implementation is a C++ library linked into user programs
Execution Overview
Execution Overview

- How is this distributed?
Execution Overview

- How is this distributed?
  1. Partition input key/value pairs into chunks, run map() tasks in parallel
Execution Overview

- How is this distributed?
  1. Partition input key/value pairs into chunks, run map() tasks in parallel
  2. After all map()s are complete, consolidate all emitted values for each unique emitted key
Execution Overview

- How is this distributed?
  1. Partition input key/value pairs into chunks, run map() tasks in parallel
  2. After all map()s are complete, consolidate all emitted values for each unique emitted key
  3. Now partition space of output map keys, and run reduce() in parallel
Execution Overview

- How is this distributed?
  1. Partition input key/value pairs into chunks, run map() tasks in parallel
  2. After all map()s are complete, consolidate all emitted values for each unique emitted key
  3. Now partition space of output map keys, and run reduce() in parallel
- If map() or reduce() fails, reexecute!
Execution in more detail
Execution in more detail

MR lib splits input. Starts master and worker processes

<table>
<thead>
<tr>
<th>Input files</th>
<th>Map phase</th>
<th>Intermediate files (on local disks)</th>
<th>Reduce phase</th>
<th>Output files</th>
</tr>
</thead>
<tbody>
<tr>
<td>split 0</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>split 1</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>split 2</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>split 3</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>split 4</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Execution in more detail

MR lib splits input.
Starts master and worker processes

Master assigns
M map tasks
R reduce tasks

Execution in more detail
Execution in more detail

MR lib splits input.
Starts master
and worker processes

Worker reads chunk
& passes <key,val>
to map function

Master assigns
M map tasks
R reduce tasks

Input files
Map phase
Intermediate files
(on local disks)
Reduce phase
Output files
Execution in more detail

MR lib splits input. Starts master and worker processes

Worker reads chunk & passes <key,val> to map function

Intermediate pairs stored in memory, written to local disk periodically.
Split using user-specified partition function. Location sent back to master

Master assigns M map tasks R reduce tasks
Execution in more detail

MR lib splits input. Starts master and worker processes

Worker reads chunk & passes <key,val> to map function

Intermediate pairs stored in memory, written to local disk periodically. Split using user-specified partition function. Location sent back to master

Master assigns M map tasks R reduce tasks

Given location from Master, Worker reads intermediate pairs, sorts by key, passes to reduce function. Result in R output files on global FS
Key Grouping
Parallel Execution

Partition function hashes by key. E.g. hash(key) mod R.
Fault Tolerance / Workers
Fault Tolerance / Workers

Task states

- idle, in-progress, completed

Handled via re-execution
Fault Tolerance / Workers

Task states
- idle, in-progress, completed

Handled via re-execution
- Detect failure via periodic heartbeats
Fault Tolerance / Workers

Task states
- idle, in-progress, completed

Handled via re-execution
- Detect failure via periodic heartbeats
- Re-execute completed + in-progress map tasks
Fault Tolerance / Workers

Task states
- idle, in-progress, completed

Handled via re-execution
- Detect failure via periodic heartbeats
- Re-execute completed + in-progress map tasks
  - Why?? (Complete tasks on local disk)
Fault Tolerance / Workers

Task states

- idle, in-progress, completed

Handled via re-execution

- Detect failure via periodic heartbeats
- Re-execute completed + in-progress map tasks
  - Why?? (Complete tasks on local disk)
- Re-execute in progress reduce tasks
Fault Tolerance / Workers

Task states
- idle, in-progress, completed

Handled via re-execution
- Detect failure via periodic heartbeats
- Re-execute completed + in-progress map tasks
  - Why?? (Complete tasks on local disk)
- Re-execute in progress reduce tasks
- Task completion committed through master
Fault Tolerance / Workers

Task states
- idle, in-progress, completed

Handled via re-execution
- Detect failure via periodic heartbeats
- Re-execute completed + in-progress map tasks
  - Why?? (Complete tasks on local disk)
- Re-execute in progress reduce tasks
- Task completion committed through master

Robust: lost 1600/1800 machines once → finished ok
Fault Tolerance / Workers

Task states
- idle, in-progress, completed

Handled via re-execution
- Detect failure via periodic heartbeats
- Re-execute completed + in-progress map tasks
  - Why?? (Complete tasks on local disk)
- Re-execute in progress reduce tasks
- Task completion committed through master

Robust: lost 1600/1800 machines once \( \rightarrow \) finished ok

Semantics in presence of failures: see paper
Master Failure

- **Could handle, ... ?**
- **But don't yet**
  - (master failure unlikely)
  - *Could use VM mechanism to hide master failure*
Refinement:

Slow workers significantly delay completion time

- Other jobs consuming resources on machine
- Bad disks w/ soft errors transfer data slowly
- Weird things: processor caches disabled (!!)

Solution: Near end of phase, spawn backup tasks

- Whichever one finishes first "wins"

Dramatically shortens job completion time
Refinement
Skipping Bad Records

- Map/Reduce functions sometimes fail for particular inputs
  - Best solution is to debug & fix
    - Not always possible ~ third-party source libraries
  - On segmentation fault:
    - Send UDP packet to master from signal handler
    - Include sequence number of record being processed
  - If master sees two failures for same record:
    - Next worker is told to skip the record
Other Refinements

- Sorting guarantees
  - within each reduce partition
- Compression of intermediate data
- Combiner
  - Useful for saving network bandwidth
- Local execution for debugging/testing
- User-defined counters
Performance

Tests run on cluster of 1800 machines:

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

Two benchmarks:

**MR_GrepScan** 1010 100-byte records to extract records matching a rare pattern (92K matching records)
Locality optimization helps:

- Input stored on FS in 64GB chunks
  - Workers are spawned near corresponding chunks
- 1800 machines read 1 TB at peak ~31 GB/s
- W/out this, rack switches would limit to 10 GB/s

Startup overhead is significant for short jobs
**MR_Sort**

- sort program sorts 1010 100-byte records (approximately 1 terabyte of data)
- map: extract 10-byte sorting key. emit key and line as value
- reduce: built-in identity function
- input data split into 64-MB pieces ($M=15000$)
- output data in 4000 files ($R=4000$)
- Partition function uses initial bytes of key to place in one of $R$ chunks
  - Local sort done for each $R$ chunk by MR before the “reduce”
  - Map task send intermediate output to local disk before shuffling to form partition
MR_Sort

Done:
839 s

Input (MB/s)

Shuffle (MB/s)

Output (MB/s)

Seconds
MR_Sort

Normal  No backup tasks  200 processes killed

```
<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Input (MB/s)</th>
<th>Shuffle (MB/s)</th>
<th>Output (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1000</td>
<td>10000</td>
<td>5000</td>
<td>20000</td>
</tr>
<tr>
<td>1000-2000</td>
<td>20000</td>
<td>10000</td>
<td>10000</td>
</tr>
</tbody>
</table>

Done: 839 s

```

```
<table>
<thead>
<tr>
<th>Time (s)</th>
<th>Input (MB/s)</th>
<th>Shuffle (MB/s)</th>
<th>Output (MB/s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0-1000</td>
<td>10000</td>
<td>5000</td>
<td>20000</td>
</tr>
<tr>
<td>1000-2000</td>
<td>20000</td>
<td>10000</td>
<td>10000</td>
</tr>
</tbody>
</table>

Done: 1235 s

```
MR_Sort

Normal

No backup tasks

200 processes killed
MR_Sort

- Backup tasks reduce job completion time a lot!
- System deals well with failures
MR_Sort

- Normal
- No backup tasks
- 200 processes killed

IO bw < grep since desired pattern uncommon
sort spends half of time writing output to local disks

- Backup tasks reduce job completion time a lot!
- System deals well with failures
MR_Sort

<table>
<thead>
<tr>
<th>Normal</th>
<th>No backup tasks</th>
<th>200 processes killed</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

IO bw < grep since desired pattern uncommon
sort spends half of time writing output to local disks

Rate data send from map tasks to reduce tasks

- Backup tasks reduce job completion time a lot!
- System deals well with failures
MR_Sort

Normal  No backup tasks  200 processes killed

- IO bw < grep since desired pattern uncommon
- sort spends half of time writing output to local disks
- Delay between first shuffling and start of writing due to sorting of intermediate data
- Rate data send from map tasks to reduce tasks

- Backup tasks reduce job completion time a lot!
- System deals well with failures
MR_Sort

- Backup tasks reduce job completion time a lot!
- System deals well with failures

Normal  No backup tasks  200 processes killed

IO bw < grep since desired pattern uncommon
sort spends half of time writing output to local disks

Delay between first shuffling and start
of writing due to sorting of intermediate
data

Rate data send from
map tasks to reduce
tasks

First hump: 1700 reduce
tasks. 1 per host out of 1700 hosts
MR_Sort

- Normal
- No backup tasks
- 200 processes killed

100 bw < grep since desired pattern uncommon
sort spends half of time writing output to local disks

Delay between first shuffling and start of writing due to sorting of intermediate data

Rate data send from map tasks to reduce tasks
First hump: 1700 reduce tasks. 1 per host out of 1700 hosts
Second hump: remaining reduce tasks

- Backup tasks reduce job completion time a lot!
- System deals well with failures
Usage in Aug 2004
### Usage in Aug 2004

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of jobs</td>
<td>29,423</td>
</tr>
<tr>
<td>Average job completion time</td>
<td>634 secs</td>
</tr>
<tr>
<td>Machine days used</td>
<td>79,186 days</td>
</tr>
</tbody>
</table>
Usage in Aug 2004

<table>
<thead>
<tr>
<th>Description</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of jobs</td>
<td>29,423</td>
</tr>
<tr>
<td>Average job completion time</td>
<td>634 secs</td>
</tr>
<tr>
<td>Machine days used</td>
<td>79,186 days</td>
</tr>
<tr>
<td>Input data read</td>
<td>3,288 TB</td>
</tr>
<tr>
<td>Intermediate data produced</td>
<td>758 TB</td>
</tr>
<tr>
<td>Output data written</td>
<td>193 TB</td>
</tr>
</tbody>
</table>
# Usage in Aug 2004

<table>
<thead>
<tr>
<th>Category</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of jobs</td>
<td>29,423</td>
</tr>
<tr>
<td>Average job completion time</td>
<td>634 secs</td>
</tr>
<tr>
<td>Machine days used</td>
<td>79,186 days</td>
</tr>
<tr>
<td>Input data read</td>
<td>3,288 TB</td>
</tr>
<tr>
<td>Intermediate data produced</td>
<td>758 TB</td>
</tr>
<tr>
<td>Output data written</td>
<td>193 TB</td>
</tr>
<tr>
<td>Average worker machines per job</td>
<td>157</td>
</tr>
<tr>
<td>Average worker deaths per job</td>
<td>1.2</td>
</tr>
</tbody>
</table>
Usage in Aug 2004

Number of jobs 29,423
Average job completion time 634 secs
Machine days used 79,186 days

Input data read 3,288 TB
Intermediate data produced 758 TB
Output data written 193 TB

Average worker machines per job 157
Average worker deaths per job 1.2
Average map tasks per job 3,351
Usage in Aug 2004

Number of jobs  29,423
Average job completion time  634 secs
Machine days used  79,186 days

Input data read  3,288 TB
Intermediate data produced  758 TB
Output data written  193 TB

Average worker machines per job  157
Average worker deaths per job  1.2
Average map tasks per job  3,351
Average reduce tasks per job  55
## Usage in Aug 2004

<table>
<thead>
<tr>
<th>Category</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of jobs</td>
<td>29,423</td>
</tr>
<tr>
<td>Average job completion time</td>
<td>634 secs</td>
</tr>
<tr>
<td>Machine days used</td>
<td>79,186 days</td>
</tr>
<tr>
<td>Input data read</td>
<td>3,288 TB</td>
</tr>
<tr>
<td>Intermediate data produced</td>
<td>758 TB</td>
</tr>
<tr>
<td>Output data written</td>
<td>193 TB</td>
</tr>
<tr>
<td>Average worker machines per job</td>
<td>157</td>
</tr>
<tr>
<td>Average worker deaths per job</td>
<td>1.2</td>
</tr>
<tr>
<td>Average map tasks per job</td>
<td>3,351</td>
</tr>
<tr>
<td>Average reduce tasks per job</td>
<td>55</td>
</tr>
</tbody>
</table>
Conclusions

- MapReduce proven to be useful abstraction
- Greatly simplifies large-scale computations
- Fun to use:
  - focus on problem,
  - let library deal w/ messy details
A giant step backward in the programming paradigm for large-scale data intensive applications

A sub-optimal implementation, in that it uses brute force instead of indexing (hash / B-trees)

Not novel at all -- it represents a specific implementation of well known techniques developed nearly 25 years ago

Missing most of the features that are routinely included in current DBMS

Incompatible with all of the tools DBMS users have come to depend on
Desktop Grids

- Use free compute, storage and network resources in Internet and Intranet environments
- Reuse existing (power, resource) infrastructure

Motivation

- High return on investment
  - Savings often a factor 5 or 10 compared to dedicated cluster
- Access to huge computational power and storage resources
State of the Art

- 400 TeraFlops/sec, over one million hosts

Loosely-coupled, single application, without time constraints

Tightly-coupled, multiple applications, with time constraints

State of the art
Challenges

- Volatility
- Resources are shared
  - Mouse/keyboard activity, user processes
  - Nondeterministic failures
    - Often 50% failure rates
- Heterogeneity
- Accessibility
  - Resources are behind NAT’s, firewalls
- Security
Outline

• BOINC
• XtremWeb
• Prediction
BOINC

- Background
  - Led by David Anderson, UC Berkeley
  - SETI@home
  - Single astronomy application
  - Too many resources

- Goals of BOINC
  - Ability to share resources among multiple projects
  - User autonomy
  - Usability
BOINC Architecture

- **BOINC Server**: Dispatcher
- **Data Repository**: Generator, Project Bookkeeping
- **File System**: Database
- **Internet**: Worker

**Processes**:
- Input file download and output upload
- Worker unit request, download
BOINC Worker Scheduling Problem

- Workers have resource share (CPU) allocation per project
- Work units per project have a deadline
- Goal: meet deadline and also resource share allocations
- Which project to schedule next on worker?
BOINC Scheduling Approach

- Use weighted round robin until a project risks missing deadline
- If so, switch to earliest deadline first scheduling
- N.B.: scheduling depends on many different parameters (e.g., availability of the resources, resource hardware, user preferences, task deadlines, resource shares, estimates of task completion time, number and characteristics of projects)
XtremWeb

- Led by Gilles Fedak (fedak@lri.fr), INRIA Futurs
- Goals
  - Support symmetric needs of users
  - Allow any node to play any role (client, worker)
  - Fault tolerance
  - Usability
XtremWeb Architecture

CLIENT

WORKER

INTERNET

PROJECT BOOKEEPING

DATABASE

FILE SYSTEM

XTREMWEB SERVER

DISPATCHER

SCHEDULER

TASK SUBMISSION

TASK REQUEST, DOWNLOAD, UPLOAD

CLIENT

WORKER

CLIENT

WORKER
Ensuring Collective Availability in Volatile Resource Pools via Forecasting

Artur Andrzejak  
Zuse-Institute Berlin (ZIB)

Derrick Kondo  
INRIA

David P. Anderson  
UC Berkeley
Motivation

- **Goal**: can we deploy *serious services / apps* over *unreliable resources*?

- **How unreliable?**
  - mostly non-dedicated PC's (used for other purposes)
  - e.g. volunteer computing Grids such as SETI@home
  - no control over availability, frequent churn

- **What are "serious" services / apps?**
  - large scale service deployment
    - examples: Amazon's EC2, TeraGrid, EGEE
  - complex applications
    - examples: DAG/message-passing applications
  - high availability: around 99.999
How to do this?

- Difficult to get (many) hosts with high avail
- Instead, we strive for *collective availability*:
  - def.: *guarantee that with high probability, in a group of \( R \geq N \) hosts, at least \( N \) remain available over time \( T \)

\( R = 6, N = 3 \)
How to do this?

- Difficult to get (many) hosts with high avail
- Instead, we strive for collective availability:
  - def.: guarantee that with high probability, in a group of $R \geq N$ hosts, at least $N$ remain available over time $T$

R = 6, N = 3
How to do this?

- Difficult to get (many) hosts with high avail
- Instead, we strive for collective availability:
  - def.: guarantee that with high probability, in a group of $R \geq N$ hosts, at least $N$ remain available over time $T$

$R = 6$, $N = 3$

$4 \geq N$ survived, col. availability achieved
Our Focus

- We use statistical and prediction methods to answer the question:
  - *Given a pool of non-dedicated hosts and a request for N hosts, how to select them such that the collective availability is maximized?*
  - i.e. at least N among R hosts "survive" interval T

- Then deployment:

  Initial prediction → Initial group → Usage over T → Which failed? → Prediction of next phase → Replacement from pool → Usage over T → Index
Availability Prediction

• We propose efficient and domain-adjusted predictions of availability for individual hosts
  – efficient:
    • fast pre-selection of predictable hosts
    • use simple and fast classification algorithm
  – domain-adjusted
    • analyze the factors of predictability and adjust our methods to them

• Then we use these individual predictions to achieve collective availability
Measurement Data

- Availability traces for over 48,000 hosts participating in SETI@home
- Active in Dec 1st, 2007 to Feb 12th, 2008
- Availability recorded by a BOINC client
  - depends whether the machine was idle
  - The definition of idle depends on user settings
- Quantized to 1 hour intervals
  - regarded as available only if *uninterrupted* avail for the *whole* hour – quite conservative
- For availability characterization, see:
Prediction Process

1. Observe Host Availability
2. Filter by High Predictability
3. Predict Individual Availability
4. Select a High-Availability Group
5. Periodically Re-Evaluate this Group
Filtering Hosts By Predictability

- We want to find out, for each host, whether its availability predictions are likely to be accurate.
- I.e. we want hosts with high **predictability**: 
  - def.: *expected accuracy of predictions from a model build on historical data*
- To estimate it, we use **indicators of predictability**
  - fast to compute (at least faster than a prediction model)
  - use only training data
Predictability Computation

- To **assess the accuracy of predictability indicators**, we have to compute for each host the true accuracy of model-based predictions.

- To this end, we **train a prediction model** on the historical availability data (4 weeks @ 1 hour), and then **compute the prediction error** on the subsequent 2 weeks (1 hour => 2*7*24 predictions).
  - This is only the "laboratory" scenario, not done in real deployment.
  - The predictability indicators should tell us, for which hosts it is not worth to build model / do predictions.
Predictability Indicators

- We have tested, among others:
  - Average length of an uninterrupted availability segment
  - Size of the compressed availability trace
    - traces with predictable patterns are likely to compress better
  - Prediction error tested on a part of the training data (as a "control indicator")
  - Number of availability state changes per week (aveSwitches)

- Evaluation:
  - correlation, scatter plots
And the winner is..

- Number of availability state changes per week: \( \text{aveSwitches} \)

<table>
<thead>
<tr>
<th>pil</th>
<th>( \text{aveAva} )</th>
<th>( \text{aveAvaRun} )</th>
<th>( \text{aveNavaRun} )</th>
<th>( \text{aveSwitches} )</th>
<th>( \text{zipPred} )</th>
<th>( \text{modelPred} )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>-0.370</td>
<td>-0.594</td>
<td>0.085</td>
<td>0.707</td>
<td>-0.654</td>
<td>-0.724</td>
</tr>
<tr>
<td>2</td>
<td>-0.275</td>
<td>-0.486</td>
<td>-0.011</td>
<td>0.678</td>
<td>-0.632</td>
<td>-0.690</td>
</tr>
<tr>
<td>4</td>
<td>-0.119</td>
<td>-0.303</td>
<td>-0.119</td>
<td>0.548</td>
<td>-0.502</td>
<td>-0.640</td>
</tr>
<tr>
<td>8</td>
<td>0.194</td>
<td>0.056</td>
<td>-0.245</td>
<td>0.195</td>
<td>-0.127</td>
<td>-0.642</td>
</tr>
<tr>
<td>16</td>
<td>0.211</td>
<td>0.091</td>
<td>-0.185</td>
<td>0.057</td>
<td>0.062</td>
<td>-0.568</td>
</tr>
</tbody>
</table>
Why are AveSwitches good?

• There are some "reasons" for data regularity → high prediction accuracy
  1. Periodic behavior, e.g. daily periodicities
  2. Long runs of availability / non-availability
  3. …

• We have studied which "reasons" are dominant:
  – by using data preprocessing which "helps" either 1 or 2

• results show that "reason" 2 is dominant

• highest accuracy for a mixture of both "reasons"
Filtering by predictability

- We create two groups for further processing:
  - low predictability, with \( \text{aveSwitches} \geq 7.47 \)
  - high predictability, with \( \text{aveSwitches} < 7.47 \)
Prediction Background

> Def. of classifier: a function which learns its output value from examples

> Function inputs are called attributes, in our study:
  > Functions of availability represented as 01 binary string
    > Time (e.g. hour in day), history bits (sum of recent k history bits)

> Output is an element from some fixed set, in our study:
  > \{0,1\} representing availability

<table>
<thead>
<tr>
<th></th>
<th>Attribute$_1$</th>
<th>...</th>
<th>Attribute$_n$</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>Example 1</td>
<td>[23,10]</td>
<td>...</td>
<td>[21,5]</td>
<td>0</td>
</tr>
<tr>
<td>Example k</td>
<td>[11,10]</td>
<td>...</td>
<td>[5,0]</td>
<td>1</td>
</tr>
<tr>
<td>Prediction</td>
<td>[1,7]</td>
<td>...</td>
<td>[13,7]</td>
<td>?</td>
</tr>
</tbody>
</table>
Prediction

• We have used a simple and fast classifier
  – naive Bayes

• The classifier takes examples i.e. vectors of measured avail + preprocessed data over 30 days

• Predicts for each hour over two weeks
  – starting now, will the host be available in the next $k$ hours
    • this is prediction interval length, $pil$

"now" prediction interval with pil = 4h
What drives accuracy?

• Dependence upon
  - prediction interval length, pil
  - training interval length
  - host ownership type (private, school, work)
Simulation Approach

- For each host in the high-predictability group make prediction at $t_0$ for pil time, and select random $R$ among those predicted as available
- $R$ depends upon:
  - $N =$ the desired number of hosts (at least $N$ should be always available)
  - the redundancy $(R-N)/N$
- Our simulations answer:
  - given $N$ and $\alpha$, the desired availability level, what is the necessary redundancy, i.e. necessary $R$?
  - a little weaker: success rate: ratio (# experiments with at least $N$ hosts alive after time $T$) / (all experiments)
Necessary Redundancy

- High predictability group (pil=4)
Necessary Redundancy

- Low predictability group (pil=4)
Is this Redundancy too high?

- In high predictability group, we have required **redundancy of 35%**
- However, we consider this dramatically low
  - In comparison, SETI@home has 200% redundancy (also used for result validation)
  - In terms of absolute savings, that equates to 165 TeraFLOPS saved in a 1 PetaFLOPS system (such as FOLDING@home) => **significant power savings**
- As a result, the BOINC consortium is interested in potentially applying our prediction schema in their job scheduling (preliminary talks)
Migration Overhead

- We also evaluated the overhead due to host migration, service restart between slices of len T
- **Threshold** = a multiple of pil which describes the total time (many T's) of running an app / service
- **Turnover rate TR**:  
  - let S be a set of hosts predicted to be available at $t_0$  
  - for those we predict which ones become not available after time pil, i.e. second prediction at $t_0+T$  
  - TR is the fraction of hosts which change from avail to non-avail  
  - essentially, the higher, the more migration needed
Turnover Rates

- about 2.5% for high predictability group
- about 12% for low predictability group
Summary

• Given that host redundancy is not an issue ("cheap" resources), high collective availability is achievable
  – even with low migration costs

• Predictability assessment and filtering is essential
  – improves accuracy
  – avoids many "wasted" predictions

• Future work:
  – hardest part: a new "application architecture" / programming model for collective availability
  – masking failures by virtualization and VM migration
References

- This work has been accepted at:

- Pdf available on request, please send e mail (derrick.kondo [at] inria.fr)
Reverse Web-Link Graph

- **Map**
  - For each URL linking to target, ...
  - Output \(<\text{target}, \text{source}>\) pairs

- **Reduce**
  - Concatenate list of all source URLs
  - Outputs: \(<\text{target}, \text{list (source)}>\) pairs
Inverted Index

- **Map ()**
  - emit <word, document ID>

- **Reduce**
  - emit <word, list (document ID)>
Job Processing

JobTracker

TaskTracker 0  TaskTracker 1  TaskTracker 2

TaskTracker 3  TaskTracker 4  TaskTracker 5
1. Client submits “grep” job, indicating code and input files
Job Processing

1. Client submits “grep” job, indicating code and input files
2. JobTracker breaks input file into k chunks, (in this case 6). Assigns work to ttrackers.
Job Processing

1. Client submits “grep” job, indicating code and input files
2. JobTracker breaks input file into k chunks, (in this case 6). Assigns work to ttrackers.
3. After map(), tasktrackers exchange map-output to build reduce() keyspace
1. Client submits “grep” job, indicating code and input files
2. JobTracker breaks input file into k chunks, (in this case 6). Assigns work to tasktrackers.
3. After map(), tasktrackers exchange map-output to build reduce() keyspace
4. JobTracker breaks reduce() keyspace into m chunks (in this case 6). Assigns work.
1. Client submits “grep” job, indicating code and input files
2. JobTracker breaks input file into k chunks, (in this case 6). Assigns work to tasktrackers.
3. After map(), tasktrackers exchange map-output to build reduce() keyspace
4. JobTracker breaks reduce() keyspace into m chunks (in this case 6). Assigns work.
5. reduce() output may go to NDFS
Task Granularity & Pipelining

- **Fine granularity tasks: map tasks >> machines**
  - Minimizes time for fault recovery
  - Can pipeline shuffling with map execution
  - Better dynamic load balancing
- Often use 200,000 map & 5000 reduce tasks
- Running on 2000 machines

| Process  | Time ------------------>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>User Program</td>
<td>MapReduce() ... wait ...</td>
</tr>
<tr>
<td>Master</td>
<td>Assign tasks to worker machines...</td>
</tr>
<tr>
<td>Worker 1</td>
<td>Map 1</td>
</tr>
<tr>
<td>Worker 2</td>
<td>Map 2</td>
</tr>
<tr>
<td>Worker 3</td>
<td>Read 1.1</td>
</tr>
<tr>
<td>Worker 4</td>
<td>Read 2.1</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>0</td>
<td>323</td>
<td>878934.6</td>
<td>1314.4</td>
<td>717.0</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>0</td>
<td>323</td>
<td>717.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>0.0</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>72.5</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>145825686</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>506631</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-operator-calls</td>
<td>508192</td>
</tr>
<tr>
<td>mr-operator-take</td>
<td>506631</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>1857</td>
<td>1707</td>
<td>878934.6</td>
<td>191995.8</td>
<td>113936.6</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>113936.6</td>
<td>57113.7</td>
<td>57113.7</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>57113.7</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>699.1</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>349.5</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>5004411944</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>17290135</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-operator-calls</td>
<td>17331371</td>
</tr>
<tr>
<td>mr-operator-outputs</td>
<td>17290135</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>5354</td>
<td>1707</td>
<td>878934.6</td>
<td>406020.1</td>
<td>241058.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>241058.2</td>
<td>196362.5</td>
<td>196362.5</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>196362.5</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>704.4</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>371.9</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>5000364228</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>17300709</td>
</tr>
<tr>
<td>dups-in-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-operator-calls</td>
<td>17342493</td>
</tr>
<tr>
<td>mr-operator-outputs</td>
<td>17300709</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>8841</td>
<td>1707</td>
<td>878934.6</td>
<td>621608.5</td>
<td>369459.8</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>369459.8</td>
<td>326986.8</td>
<td>326986.8</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>0</td>
<td>326986.8</td>
<td>0.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>706.5</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>419.2</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>4982870667</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>17229926</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-operator-calls</td>
<td>17272056</td>
</tr>
<tr>
<td>mr-operator-outputs</td>
<td>17229926</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>195</td>
<td>305</td>
<td>523499.2</td>
<td>523389.6</td>
<td>523389.6</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>195</td>
<td>523389.6</td>
<td>2685.2</td>
<td>2742.6</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.3</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.5</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>45.7</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>2313178</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>7936</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>1954105</td>
</tr>
<tr>
<td>mr-merge-outputs</td>
<td>1954105</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>523499.5</td>
<td>523499.5</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>523499.5</td>
<td>133837.8</td>
<td>136929.6</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.1</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>1238.8</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>0.10</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>0</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>51738599</td>
</tr>
<tr>
<td>mr-merge-outputs</td>
<td>51738599</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>523499.5</td>
<td>523499.5</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>523499.5</td>
<td>263283.3</td>
<td>269351.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Counters</th>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td></td>
<td>Output (MB/s)</td>
<td>1225.1</td>
</tr>
<tr>
<td></td>
<td>doc-index-hits</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>docs-indexed</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>mr-merge-calls</td>
<td>51842100</td>
</tr>
<tr>
<td></td>
<td>mr-merge-outputs</td>
<td>51842100</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>523499.5</td>
<td>523499.5</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>0</td>
<td>500</td>
<td>523499.5</td>
<td>390447.6</td>
<td>399457.2</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>1222.0</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>0.016</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>0</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>51640600</td>
</tr>
<tr>
<td>mr-merge-outputs</td>
<td>51640600</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>520468.6</td>
<td>520468.6</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>406</td>
<td>94</td>
<td>520468.6</td>
<td>512265.2</td>
<td>514373.3</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>849.5</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>0</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>0</td>
</tr>
<tr>
<td>dups-in-indexed</td>
<td>0</td>
</tr>
<tr>
<td>mmerge-merge-calls</td>
<td>35083350</td>
</tr>
<tr>
<td>mmerge-outputs</td>
<td>35083350</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Map</strong></td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td><strong>Shuffle</strong></td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>519781.8</td>
<td>519781.8</td>
</tr>
<tr>
<td><strong>Reduce</strong></td>
<td>500</td>
<td>498</td>
<td>2</td>
<td>519781.8</td>
<td>519394.7</td>
<td>519440.7</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>9.4</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>0.0</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>0.0</td>
</tr>
<tr>
<td>dups-in-index</td>
<td>0.0</td>
</tr>
<tr>
<td>merge-calls</td>
<td>394792</td>
</tr>
<tr>
<td>merge-outputs</td>
<td>394792</td>
</tr>
</tbody>
</table>
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

<table>
<thead>
<tr>
<th>Type</th>
<th>Shards</th>
<th>Done</th>
<th>Active</th>
<th>Input(MB)</th>
<th>Done(MB)</th>
<th>Output(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Map</td>
<td>13853</td>
<td>13853</td>
<td>0</td>
<td>878934.6</td>
<td>878934.6</td>
<td>523499.2</td>
</tr>
<tr>
<td>Shuffle</td>
<td>500</td>
<td>500</td>
<td>0</td>
<td>523499.2</td>
<td>519774.3</td>
<td>519774.3</td>
</tr>
<tr>
<td>Reduce</td>
<td>500</td>
<td>499</td>
<td>1</td>
<td>519774.3</td>
<td>519735.2</td>
<td>519764.0</td>
</tr>
</tbody>
</table>

Counters

<table>
<thead>
<tr>
<th>Variable</th>
<th>Minute</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mapped (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Shuffle (MB/s)</td>
<td>0.0</td>
</tr>
<tr>
<td>Output (MB/s)</td>
<td>1.9</td>
</tr>
<tr>
<td>doc-index-hits</td>
<td>0 105</td>
</tr>
<tr>
<td>docs-indexed</td>
<td>0 105</td>
</tr>
<tr>
<td>dups-in-index-merge</td>
<td>0 0</td>
</tr>
<tr>
<td>mr-merge-calls</td>
<td>73442</td>
</tr>
<tr>
<td>mr-merge-outputs</td>
<td>73442</td>
</tr>
</tbody>
</table>
Refinement:

- **Master scheduling policy:**
  - Asks GFS for locations of replicas of input file blocks
  - Map tasks typically split into 64MB (GFS block size)
  - Map tasks scheduled so GFS input block replica are on same machine or same rack

- **Effect**
  - Thousands of machines read input at local disk speed
    - Without this, rack switches limit read rate
EC2 SOAP/Query API

- **Images:**
  - RegisterImage
  - DescribeImages
  - DeregisterImage

- **Instances:**
  - RunInstances
  - DescribeInstances
  - TerminateInstances
  - GetConsoleOutput
  - RebootInstances

- **Keypairs:**
  - CreateKeyPair
  - DescribeKeyPairs
  - DeleteKeyPair

- **Image Attributes:**
  - ModifyImageAttribute
  - DescribeImageAttribute
  - ResetImageAttribute

- **Security Groups:**
  - CreateSecurityGroup
  - DescribeSecurityGroups
  - DeleteSecurityGroup
  - AuthorizeSecurityGroupIngress
  - RevokeSecurityGroupIngress
CloudFront
Experience

Rewrote Google's production indexing System using MapReduce

- Set of 10, 14, 17, 21, 24 MapReduce operations
- New code is simpler, easier to understand
  - 3800 lines C++ → 700
- MapReduce handles failures, slow machines
- Easy to make indexing faster
Related Work

- Programming model inspired by functional language primitives
- Partitioning/shuffling similar to many large-scale sorting systems
  - NOW-Sort ['97]
- Re-execution for fault tolerance
  - BAD-FS ['04] and TACC ['97]
- Locality optimization has parallels with Active Disks/Diamond work
  - Active Disks ['01], Diamond ['04]
- Backup tasks similar to Eager Scheduling in Charlotte system
  - Charlotte ['96]
- Dynamic load balancing solves similar problem as
Cloud versus the Grid

- Geographically distributed
- Across multiple administrative domains
- App’s need high-level programming abstractions (e.g. workflow)
Steps

- **Get Amazon account**
  - [http://www.amazonaws.com](http://www.amazonaws.com)
- Boot instance of AMI image
- Log in with ssh
- Start Apache