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 & contrast with clouds
  - Architecture
  - Prediction

Motivation

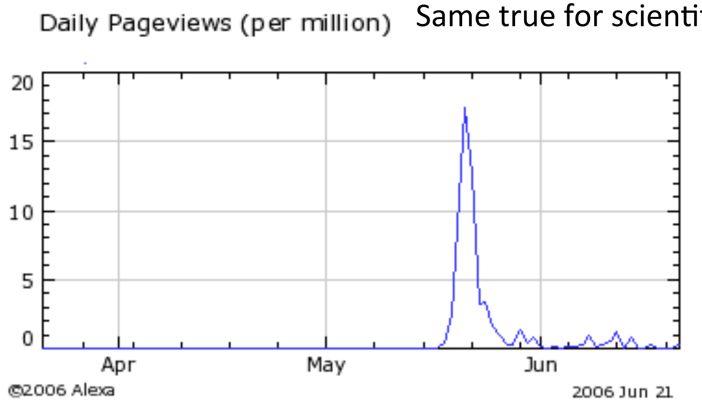
- 70% of Web Development Effort is “Muck”:
  - Data Centers
  - Bandwidth / Power / Cooling
  - Operations
  - Staffing
- Scaling is Difficult and Expensive:
  - Large Up-Front Investment
  - Invest Ahead of Demand
  - Load is Unpredictable

operations = billing

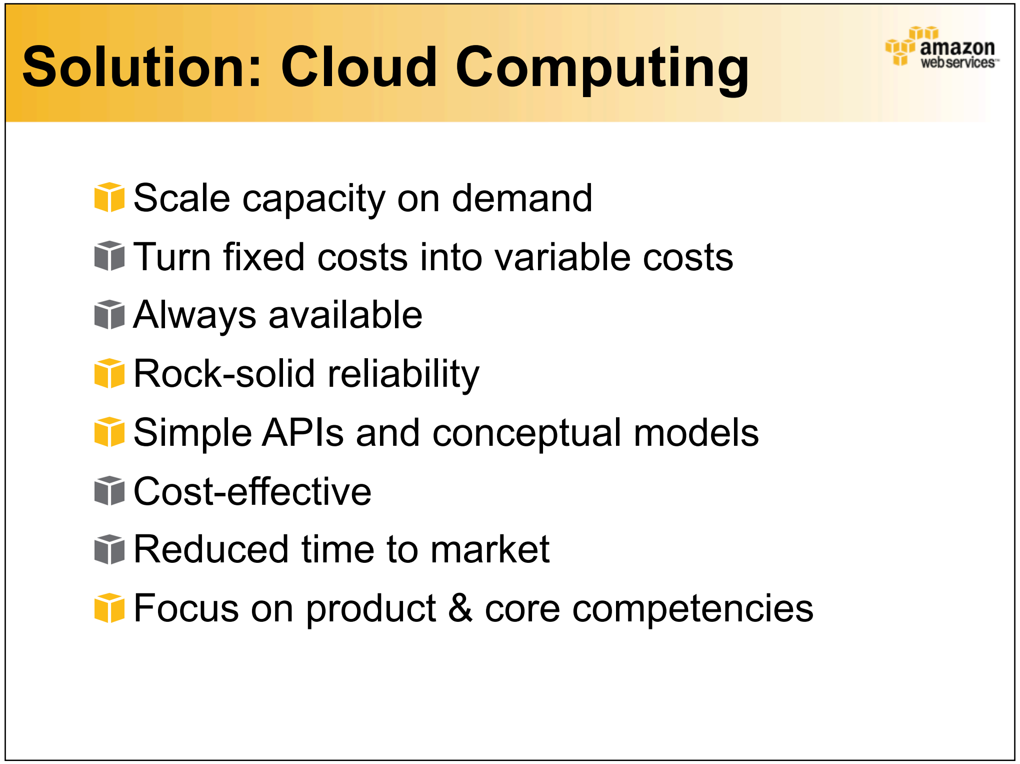
Dream or Nightmare?

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

Same true for scientific workloads



same true for scientific workloads



AWS will use commercially reasonable efforts to make Amazon EC2 available with an Annual Uptime Percentage (defined below) of at least 99.95% during the Service Year. In the event Amazon EC2 does not meet the Annual Uptime Percentage commitment, you will be eligible to receive a Service Credit as described below.

Service Commitments and Service Credits

If the Annual Uptime Percentage for a customer drops below 99.95% for the Service Year, that customer is eligible to receive a Service Credit equal to 10% of their bill (excluding one-time payments made for Reserved Instances) for the Eligible Credit Period. To file a claim, a customer does not have to have wait 365 days from the day they started using the service or 365 days from their last successful claim. A customer can file a claim any time their Annual Uptime Percentage over the trailing 365 days drops below 99.95%.

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 åand consumers." -- Prof Raj Buyya

globus : push model for accessing resources

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

Figure 5: Sun Microsystems Black Box

1,152 systems in 20x8x8 foot container

from hamilton paper:
container to reduce shipping, packaging, and deployment costs

racks consist of 2-way SMP's

Types of Clouds

- Platform-as-a-service
  - E.g. Amazon's EC2
- Applications-as-a-service
  - E.g. Google App Engine (DataStore/GQL, MapReduce)

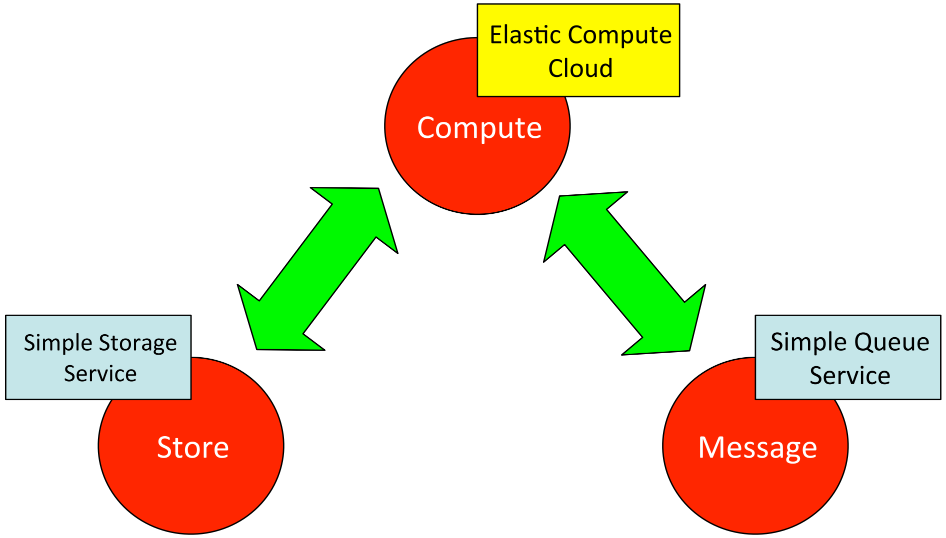
While others allow for the installation and configuration of nearly any *NIX compatible software, AppEngine requires developers to use Python as the programming language and "Datastore" – a version of Google's proprietary BigTable – for data persistence.

App Engine can only execute code called from an HTTP request.

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



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

In computer security, an access control list (ACL) is a list of permissions attached to an object. The list specifies who or what is allowed to access the object and what operations are allowed to be performed on the object. In a typical ACL, each entry in the list specifies a subject and an operation: for example, the entry (Alice, delete) on the ACL for file WXY gives Alice permission to delete file WXY.

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***

internal transfers free

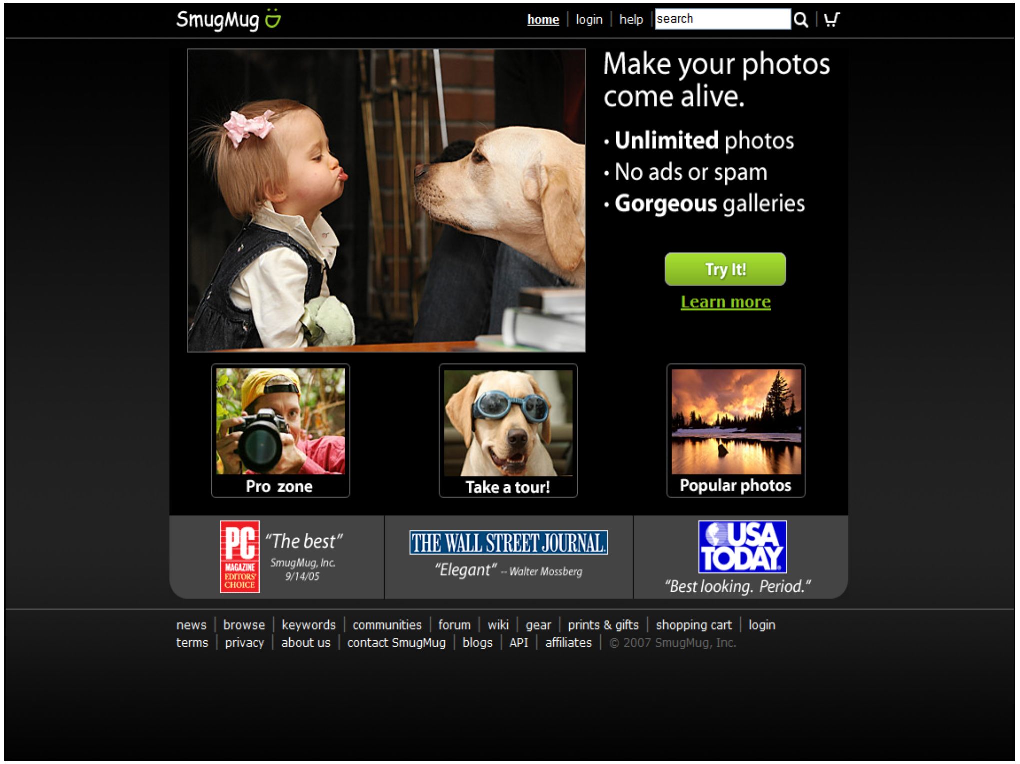
Amazon S3 Concepts

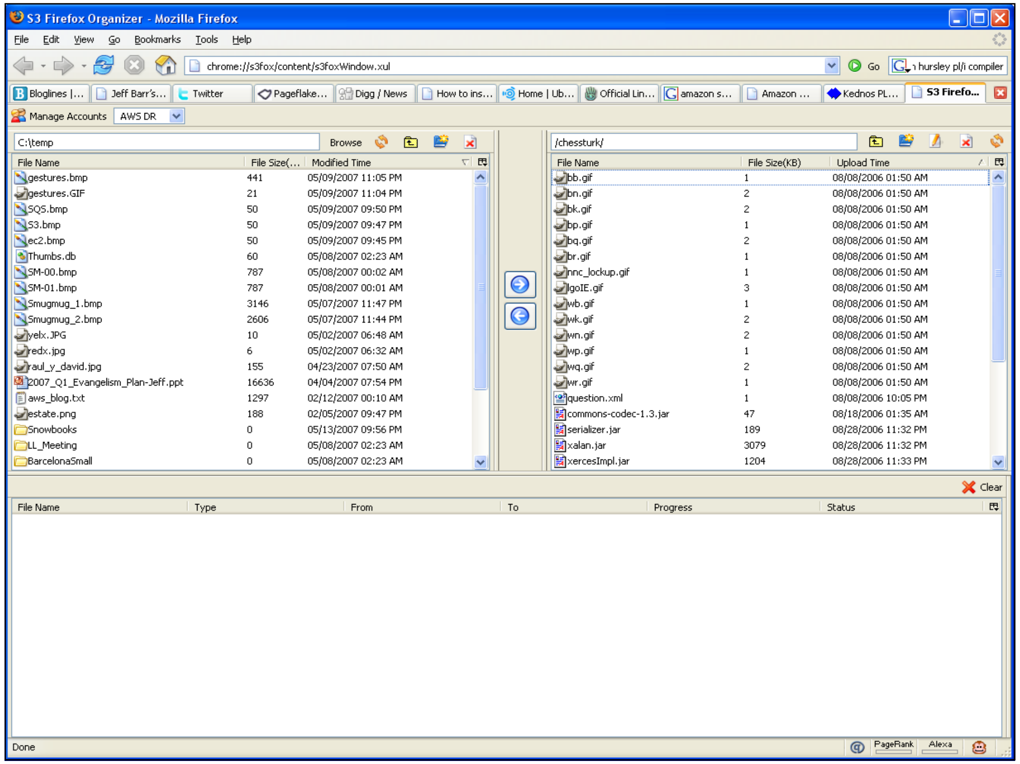
- Objects:
  - Opaque data to be stored (1 byte … 5 Gigabytes)
  - Metadata (attribute-value, up to 4KB)
  - Authentication and access controls
- Buckets (like directories):
  - Object container – any number of objects
  - 100 buckets per account / buckets are “owned”
- Keys:
  - Unique object identifier 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

REST strictly refers to a collection of network architecture principles which outline how resources are defined and addressed. The term is often used in a looser sense to describe any simple interface which transmits domain-specific data over HTTP without an additional messaging layer such as SOAP or session tracking via HTTP cookies.

SOAP, originally defined as Simple Object Access Protocol, is a protocol specification for exchanging structured information in the implementation of Web Services in computer networks. It relies on Extensible Markup Language (XML) as its message format and usually relies on other Application Layer protocols, most notably Remote Procedure Call (RPC) and HTTP for message negotiation and transmission. SOAP forms the foundation layer of the web services protocol stack providing a basic messaging framework upon which abstract layers can be built.


SmugMug
home | login | help | search
Make your photos come alive.
• Unlimited photos
• No ads or spam
• Gorgeous galleries
Try It!
Learn more
Pro zone
Take a tour!
Popular photos
PC MAGAZINE EDITORS' CHOICE
"The best"
SmugMug, Inc.
9/14/05
THE WALL STREET JOURNAL.
"Elegant" -- Walter Mossberg
USA TODAY
"Best looking. Period."
news | browse | keywords | communities | forum | wiki | gear | prints & gifts | shopping cart | login
terms | privacy | about us | contact SmugMug | blogs | API | affiliates | © 2007 SmugMug, Inc.


S3 Firefox Organizer - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
chrome://s3fox/content/s3foxWindow.xul
Go
Bloglines | ...
Jeff Barr's...
Twitter
Pageflake...
Digg / News
How to ins...
Home | Ub...
Official Lin...
amazon s...
Amazon ...
Kednos PL...
S3 Firefo...
Manage Accounts
AWS DR
C:\temp
Browse
File Name
File Size(...
Modified Time
gestures.bmp 441 05/09/2007 11:05 PM
gestures.GIF 21 05/09/2007 11:04 PM
SQS.bmp 50 05/09/2007 09:50 PM
S3.bmp 50 05/09/2007 09:47 PM
ec2.bmp 50 05/09/2007 09:45 PM
Thumbs.db 60 05/08/2007 02:23 AM
SM-00.bmp 787 05/08/2007 00:02 AM
SM-01.bmp 787 05/08/2007 00:01 AM
Smugmug_1.bmp 3146 05/07/2007 11:47 PM
Smugmug_2.bmp 2606 05/07/2007 11:44 PM
yelx.JPG 10 05/02/2007 06:48 AM
redx.jpg 6 05/02/2007 06:32 AM
raul_y_david.jpg 155 04/23/2007 07:50 AM
2007_Q1_Evangelism_Plan-Jeff.ppt 16636 04/04/2007 07:54 PM
aws_blog.txt 1297 02/12/2007 00:10 AM
estate.png 188 02/05/2007 09:47 PM
Snowbooks 0 05/13/2007 09:56 PM
LL_Meeting 0 05/08/2007 02:23 AM
BarcelonaSmall 0 05/08/2007 02:23 AM
/chessturk/
File Name
File Size(KB)
Upload Time
bb.gif 1 08/08/2006 01:50 AM
bn.gif 2 08/08/2006 01:50 AM
bk.gif 2 08/08/2006 01:50 AM
bp.gif 1 08/08/2006 01:50 AM
bq.gif 2 08/08/2006 01:50 AM
br.gif 1 08/08/2006 01:50 AM
nnc_lockup.gif 1 08/08/2006 01:50 AM
lgoIE.gif 3 08/08/2006 01:50 AM
wb.gif 1 08/08/2006 01:50 AM
wk.gif 2 08/08/2006 01:50 AM
wn.gif 2 08/08/2006 01:50 AM
wp.gif 1 08/08/2006 01:50 AM
wq.gif 2 08/08/2006 01:50 AM
wr.gif 1 08/08/2006 01:50 AM
question.xml 1 08/08/2006 10:05 PM
commons-codec-1.3.jar 47 08/18/2006 01:35 AM
serializer.jar 189 08/28/2006 11:32 PM
xalan.jar 3079 08/28/2006 11:32 PM
xercesImpl.jar 1204 08/28/2006 11:33 PM
Clear
File Name
Type
From
To
Progress
Status
Done
PageRank
Alexa

Amazon Elastic Compute Cloud

EC2

Amazon EC2

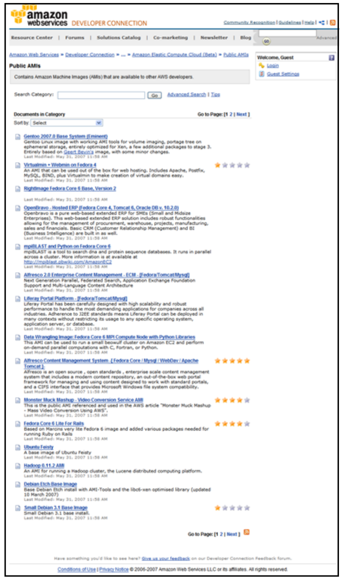
- Virtual environment for linux/windows applications
  - Create Amazon Machine Image (AMI) with app's, lib's, data, config settings,
  - Upload image to S3, then start/stop/monitor images

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
  - Persistant 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.

Large instances

- 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

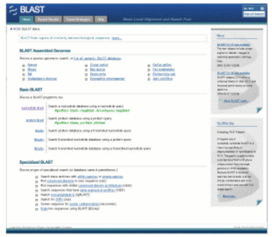
- Operating Systems
  - Red Hat Enterprise Linux Windows Server 2003 Oracle Enterprise Linux
  - OpenSolaris openSUSE Linux Ubuntu Linux
  - Fedora Gentoo Linux Debian
- Software
  - Databases
    - Oracle 11g, MySQL Enterprise, Microsoft SQL Server Standard 2005
  - Batch Processing
    - Hadoop, Condor
  - eb Hosting
    - Apache HTTP, IIS/Asp.Net

Pricing

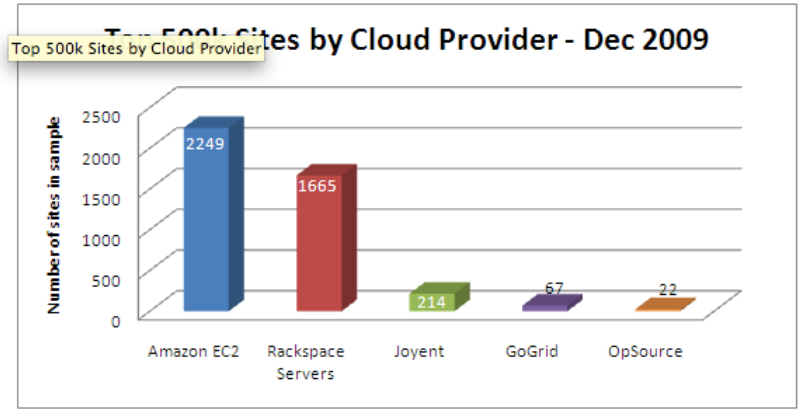
- Pay as you use
- Standard Instances
  - Linux
    - Small (Default) $0.10 per hour
    - Large $0.40 per hour
    - Extra Large $0.80 per hour
- High CPU Instances
  - Medium $0.20 per hour
  - Extra Large $0.80 per hour
- Internet Data Transfer
  - Data transfer in: $0.10 per GB
  - Data transfer out: $0.17 per GB

Amazon EC2 At Work

- Startups
  - Cruxy – Media transcoding
  - 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



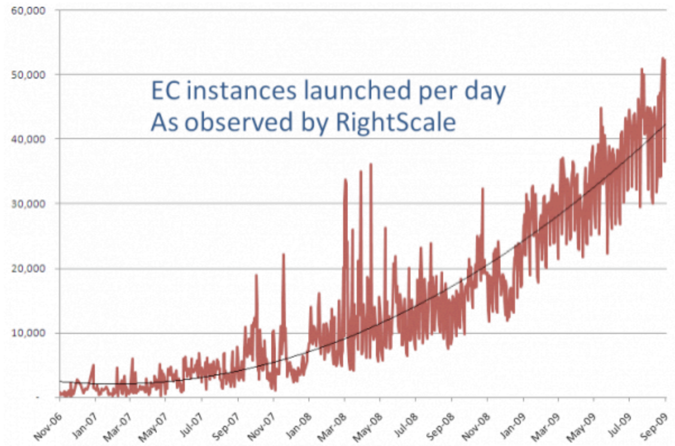
Usage



http://www.jackofallclouds.com/2009/12/state-of-the-cloud-december-2009/

The input dataset was QuantCast's top site index. We took the top 500,000 sites listed and ran them through our scanning tools to build an index of the websites which are hosted on Amazon EC2. So, how many of the world's top websites are placing their gateway to the world and, in many cases, their entire business – in the hands of Amazon's cloud?

Usage



http://www.jackofallclouds.com/2009/10/amazon-usage-estimates-and-updates/

220m annual revenue. http://cloudscaling.com/blog/cloud-computing/amazons-ec2-generating-220m-annually

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 2. The resources needed to provide high performance data access, high data availability and long data durability are different

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

Table 3. Application classes and their associated requirements

Application class	Durability	Availability	High access speed
Cache	No	Depends	Yes
Long-term archival	Yes	No	No
Online production	No	Yes	Yes
Batch production	No	No	Yes

First, we note that, for batch processing with little or no interactive user input, the relatively slow access S3 provides to individual data items does not have a significant impact on user observed performance as long as jobs are specified in advance and S3 is able to provide data at an overall rate faster than the rate at which it can be processed. To this end, an efficient system using

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

* A cluster is a distributed system that consists of a number of identical machines owned by a single entity, usually stacked up in a closet or a machine room. Clusters became the most common form of high performance computing in the 1990s and are the type of system now dominating the Top 500 List of supercomputers.

* A grid is a distributed system that enables people to access computing resources from different institutions over the wide area. The term grid computing was coined by Ian Foster and Carl Kesselman in the late 1990s to describe easy access to large scale computational power. Examples of grids include TeraGrid and the Open Science Grid.

* A cloud is a distributed system where the user doesn't care exactly what resources are used to carry out an operation; this is virtualization in the most abstract sense. There exist commercial clouds such as Amazon EC2, as well proprietary clouds found in nearly any industrial scale web site.

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 **f list [list$_2$ list$_3$ …]**)

Unary operator

- (map square '(1 2 3 4))
  - (1 4 9 16)

Binary operator

-))
  - (+ 16 (+ 9 (+ 4 1)))
  - 30

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

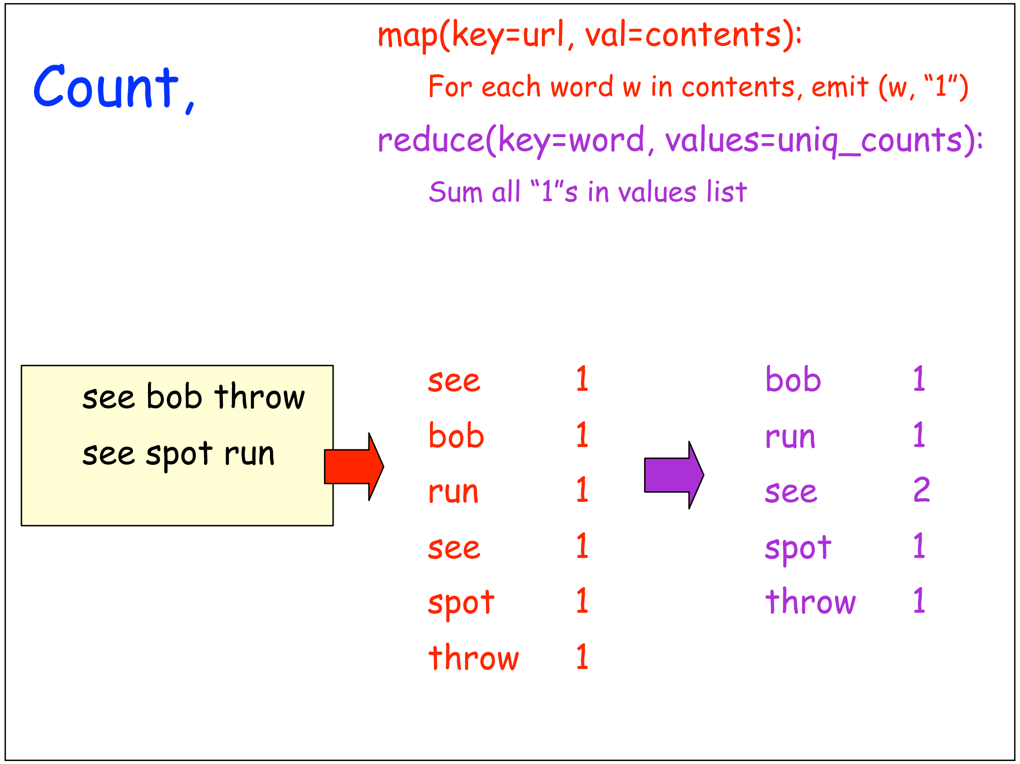
map (k1,v1) → list(k2,v2)
reduce (k2,list(v2)) → list(v2)

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)”



input set: set of words in document specified by url

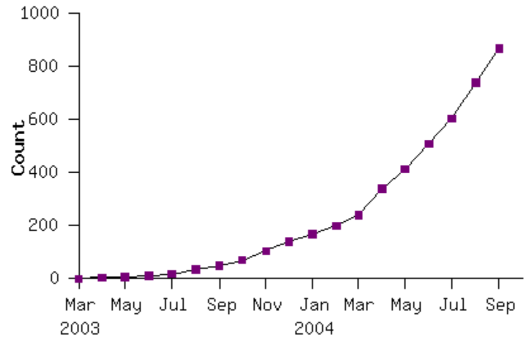
Grep

- 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

input set: set of lines

Offset specifies line in document

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

Reverse Web-Link Graph: The map function outputs ⟨target, source⟩ pairs for each link to a target URL found in a page named source. The reduce function concatenates the list of all source URLs associated with a given target URL and emits the pair: target, l ist(source)

Term-Vector per Host: A term vector summarizes the most important words that occur in a document or a set of documents as a list of ⟨word, f requency⟩ pairs. The map function emits a ⟨hostname, term vector⟩ pair for each input document (where the hostname is extracted from the URL of the document). The reduce function is passed all per-document term vectors for a given host. It adds these term vectors together, throwing away infrequent terms, and then emits a final hostname, term vector pair.

InvertedIndex: Themapfunctionparseseachdocument, andemitsasequenceof ⟨word,documentID⟩ pairs. Thereducefunctionacceptsall pairsforagiven word,sortsthecorrespondingdocumentIDsandemitsa ⟨word,list(documentID)⟩pair. Thesetofalloutput pairsformsasimpleinvertedindex. Itiseasytoaugment thiscomputationtokeeptrackofwordpositions.

DistributedSort: Themapfunctionextractsthekey fromeachrecord,andemitsa ⟨key,record⟩pair. The reducefunctionemitsallpairsunchanged. Thiscomputationdependsonthepartitioningfacilitiesdescribedin Section4.1andtheorderingpropertiesdescribedinSection4.2.

Implementation Overview

Typical cluster:

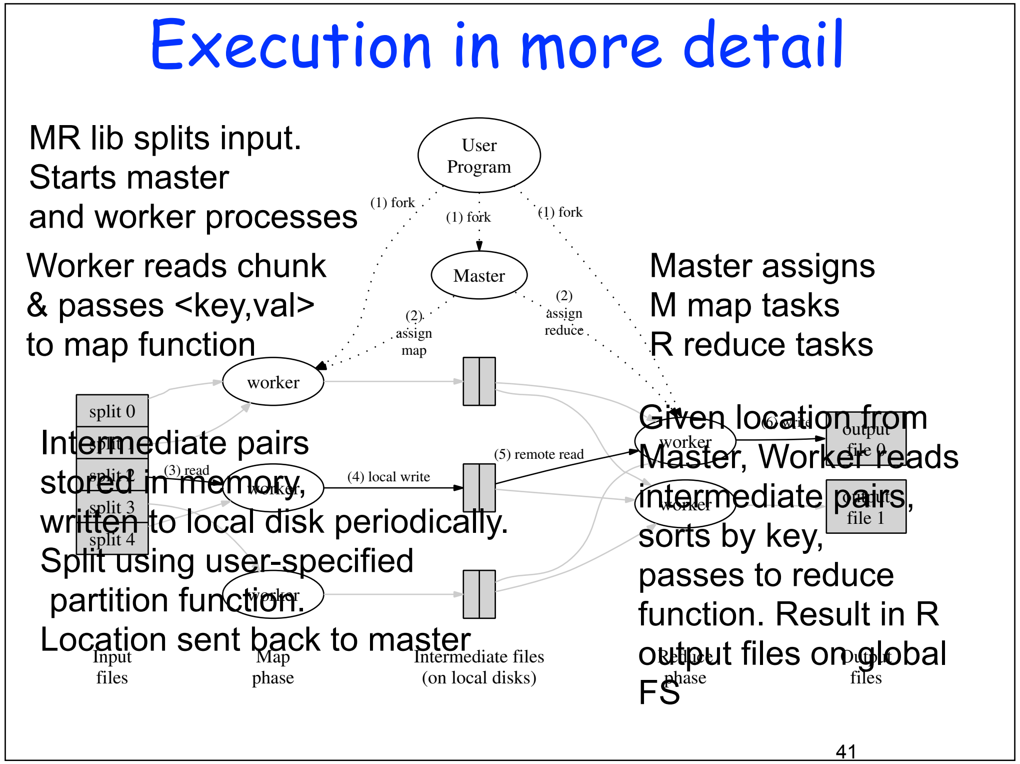
- 100s/1000s of 2-CPU x86 machines, 2-4 GB of memory
- Limited bisection bandwidth
- Storage is on local IDE disks
- GFS: distributed file system manages data (SOSP'03)
- Job scheduling system: jobs made up of tasks,
  scheduler assigns tasks to machines

Implementation is a C++ library linked into user programs

Execution Overview

- 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!

16-64MB chuncks



num reduce tasks dependent on split function set by user

http://www.databasecolumn.com/2008/01/mapreduce-a-major-step-back.html

The basic idea of MapReduce is straightforward. It consists of two programs that the user writes called map and reduce plus a framework for executing a possibly large number of instances of each program on a compute cluster.

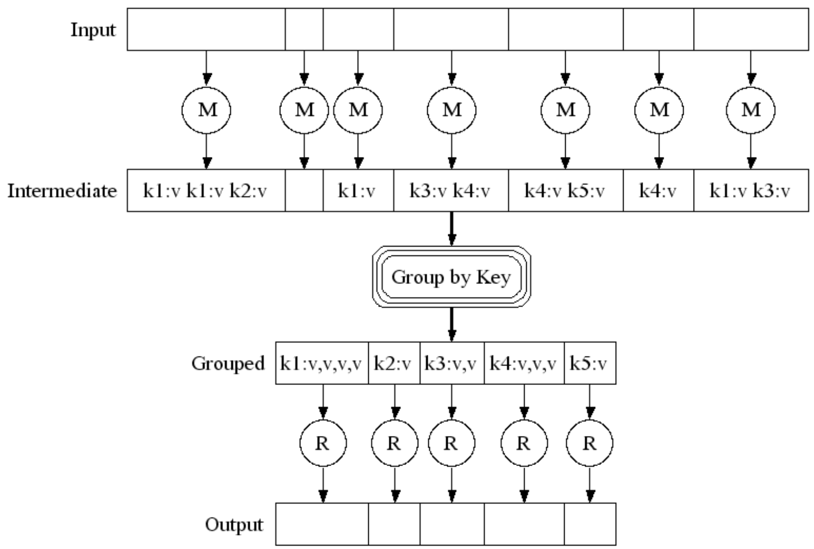
The map program reads a set of "records" from an input file, does any desired filtering and/or transformations, and then outputs a set of records of the form (key, data). As the map program produces output records, a "split" function partitions the records into M disjoint buckets by applying a function to the key of each output record.   This split function is typically a hash function, though any deterministic function will suffice. When a bucket fills, it is written to disk. The map program terminates with M output files, one for each bucket.

In general, there are multiple instances of the map program running on different nodes of a compute cluster. Each map instance is given a distinct portion of the input file by the MapReduce scheduler to process. If N nodes participate in the map phase, then there are M files on disk storage at each of N nodes, for a total of N * M files; Fi,j, 1 ≤ i ≤ N, 1 ≤ j ≤ M.

The key thing to observe is that all map instances use the same hash function. Hence, all output records with the same hash value will be in corresponding output files.

The second phase of a MapReduce job executes M instances of the reduce program, Rj, 1 ≤ j ≤ M. The input for each reduce instance Rj consists of the files Fi,j, 1 ≤ i ≤ N. Again notice that all output records from the map phase with the same hash value will be consumed by the same reduce instance -- no matter which map instance produced them. After being collected by the map-reduce framework, the input records to a reduce instance are grouped on their keys (by sorting or hashing) and feed to the reduce program. Like the map program, the reduce program is an arbitrary computation in a general-purpose language. Hence, it can do anything it wants with its records. For example, it might compute some additional function over other data fields in the record. Each reduce instance can write records to an output file, which forms part of the "answer" to a MapReduce computation.

Key Grouping



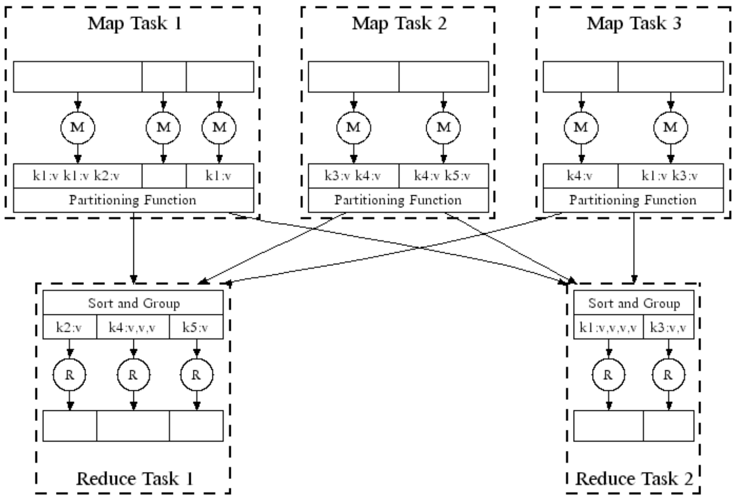
Better to call reduce function once for same key. (E.g. in word counting, <bob, (1, 1, 1, 1, 1)> versus multiple calls to <bob, 1>, <bob, 1>, <bob, 1>

DPK: function of the combiner

E.g. word count. Will have many “the”, 1 key value pairs

Parallel Execution

Partition function hashes by key. E.g. hash(key) mod R.



Partition function ensures pairs with the same key are within the same reduce task???? But intermediate output on local disk??

http://en.wikipedia.org/wiki/MapReduce

Partition function

The output of all of the maps is allocated to a particular *reducer* by the application's *partition* function. The *partition* function is given the key and the number of reducers and returns the index of the desired *reduce*.

A typical default is to hash the key and modulo the number of *reducers*.

[edit]

Comparison function

The input for each *reduce* is pulled from the machine where the *map* ran and sorted using the application's *comparison* function.

[edit]

Reduce function

The framework calls the application's *reduce* function once for each unique key in the sorted order. The *reduce* can iterate through the values that are associated with that key and output 0 or more values.

In the word count example, the *reduce* function takes the input values, sums them and generates a single output of the word and the final sum.

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

Semantics in presence of failures: see paper

Completed map tasks are re-executed on a failure because their output is stored on the local disk(s) of the failed machine and is therefore inaccessible. Completed reduce tasks do not need to be re-executed since their output is stored in a global file system.

Completed map tasks are re-executed on a failure because their output is stored on the local disk(s) of the failed machine and is therefore inaccessible. Completed reduce tasks do not need to be re-executed since their output is stored in a global file system.

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

http://www.databasecolumn.com/2008/01/mapreduce-a-major-step-back.html

locality:
store files on FGS in 64MB blocks

We guarantee that within a given partition, the intermediate key/value pairs are processed in increasing key order. This ordering guarantee makes it easy to generate a sorted output file per partition, which is useful when the output file format needs to support efficient random access lookups by key, or users of the output find it convenient to have the data sorted.

combine - reduces on local disk of map task before sending info over network. E.g. word count task with repetitive works

The MapReduce library provides a counter facility to count occurrences of various events. For example, user code may want to count total number of words processed or the number of German documents indexed, etc.

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)

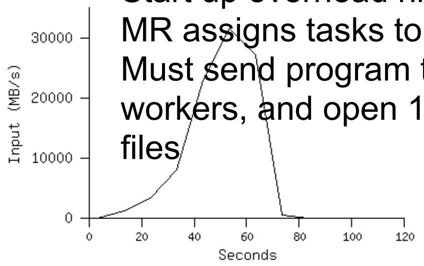
Jump to: navigation, search

If the network is segmented into two equal parts, this is the bandwidth between the two parts.[1] Typically, this refers to the worst-case segmentation, but being of equal parts is critical to the definition, as it refers to an actual bisection of the network.

Bisection bandwidth. The bidirectional capacity of a network between two equal-sized partitions of nodes. The cut across the network is taken at the narrowest point in each bisection.

MR_Grep

Rate at which input is scanned



Start up overhead high as MR assigns tasks to workers. Must send program to all workers, and open 1000 input files

Time

Locality optimization helps:

- Input stored on FS in 64GB chunks
  - Workers are spawned near corresponding chucks
- 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

y-axis: rate at which input is scanned

startup overhead high due as MR assigned tasks to workers

This includes about a minute of startup overhead. The overhead is due to the propagation of the program to all worker machines, and delays interacting with GFS to open the set of 1000 input files and to get the information needed for the locality optimization.

Network bandwidth is a relatively scarce resource in our computing environment. We conserve network bandwidth by taking advantage of the fact that the input data (managed by GFS [8]) is stored on the local disks of the machines that make up our cluster. GFS divides each file into 64 MB blocks, and stores several copies of each block (typically 3 copies) on different machines. The MapReduce master takes the location information of the input files into account and attempts to schedule a map task on a machine that contains a replica of the corresponding input data. Failing that, it attempts to schedule a map task near a replica of that task's input data (e.g., on a worker machine that is on the same network switch as the machine containing the data). When running large MapReduce operations on a significant fraction of the workers in a cluster, most input data is read locally and consumes no network bandwidth.

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

map-reduce-review.pdf

The MapReduce sort has been written in roughly 50 lines of code which is impressive given the complexity of the parallelization of the computation. MapReduce already processes keys in a sorted order, so most of the work has already been done in the MapReduce infrastructure. As with NOW Sort, there are 100 byte records and 10 byte keys. The keys are then separated from the records. In order to partition inputs across the workers, the modulo function is used by looking at the top few bits (dpk: highest bits) and finding the remainder (of those highest bits) when divided by the number of mappers. In this case, a reducer is not really needed since the data is already sorted by the end of the map phase. The reducer then is simply the identity function.

Both the NOW sort and the MapReduce sort get the input data from the local disk initially. Also, both methods assume an initial even distribution of keys across all the nodes. As NOWSort reads the keys, it sends each key to the correct machine with the appropriate bucket (which could be remote). However, MapReduce sends to the local disk first before the shuffle phase begins. The final phase for both methods is simply a local sort of the data. NOWSort finishes by writing the output data to the local disk. MapReduce on the other hand uses GFS to store output data, so one replica is produced. This means that two writes are needed at the end which decreases performance but is more robust to failure.

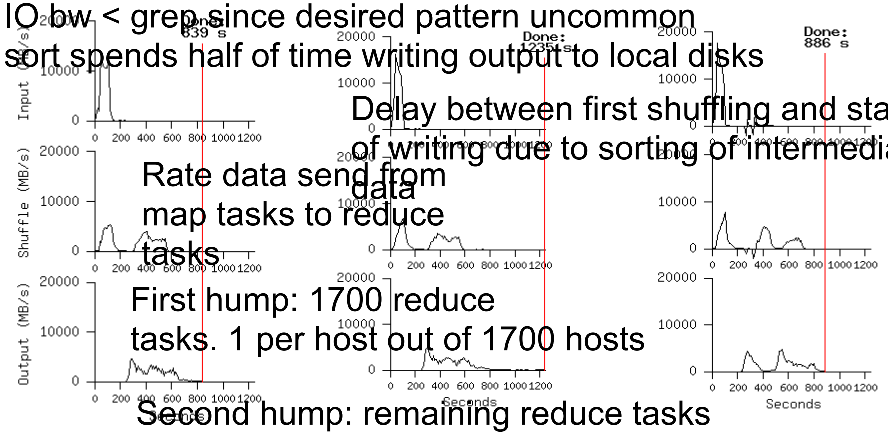
2

Ourpartitioningfunctionforthisbenchmarkhasbuilt-inknowledgeof thedistributionof keys. Inageneral sortingprogram, wewouldaddapre-passMapReduce operationthat wouldcollect asampleof thekeysand usethedistributionofthesampledkeystocomputesplit-pointsforthefinalsortingpass.

there is a final sorting pass

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

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

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	

Conclusions

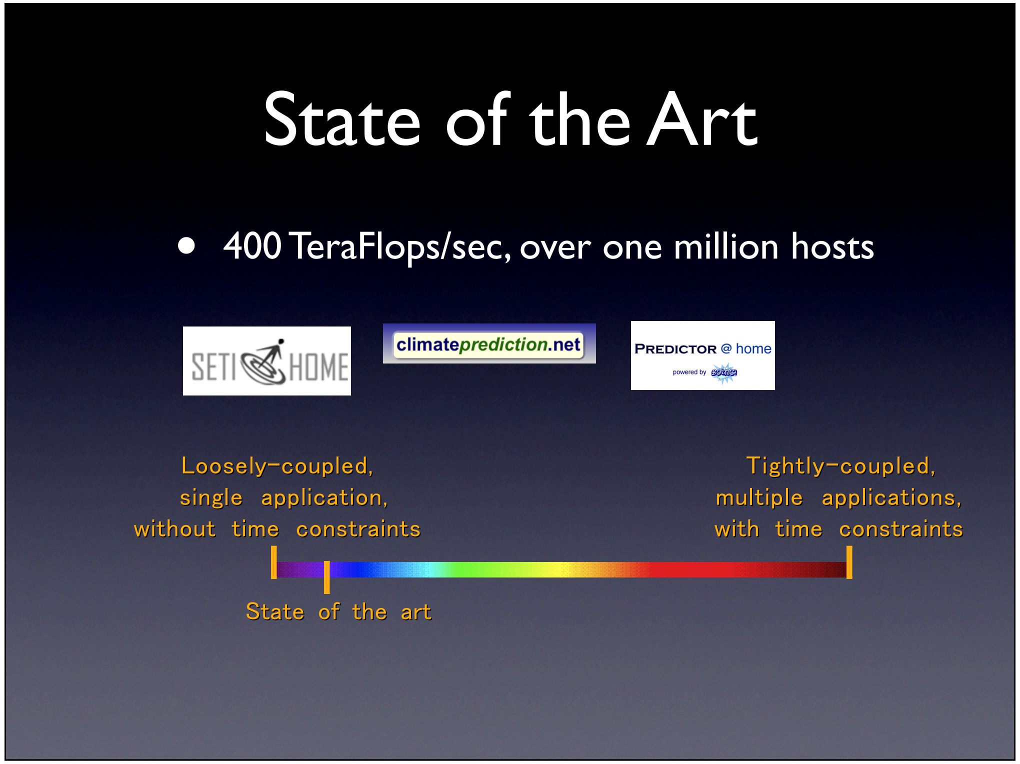
- MapReduce proven to be useful abstraction
- Greatly simplifies large-scale computations
- Fun to use:
  - focus on problem,
  - let library deal w/ messy details

A major step backwards

- http://www.databasecolumn.com/2008/01/mapreduce-a-major-step-back.html
- 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



The future looks promising with the rapid improvement of commodity components. Gigabit switches, fiber optic networks, multi-core processors.

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

Use of resources across organizational domains through virtual organization

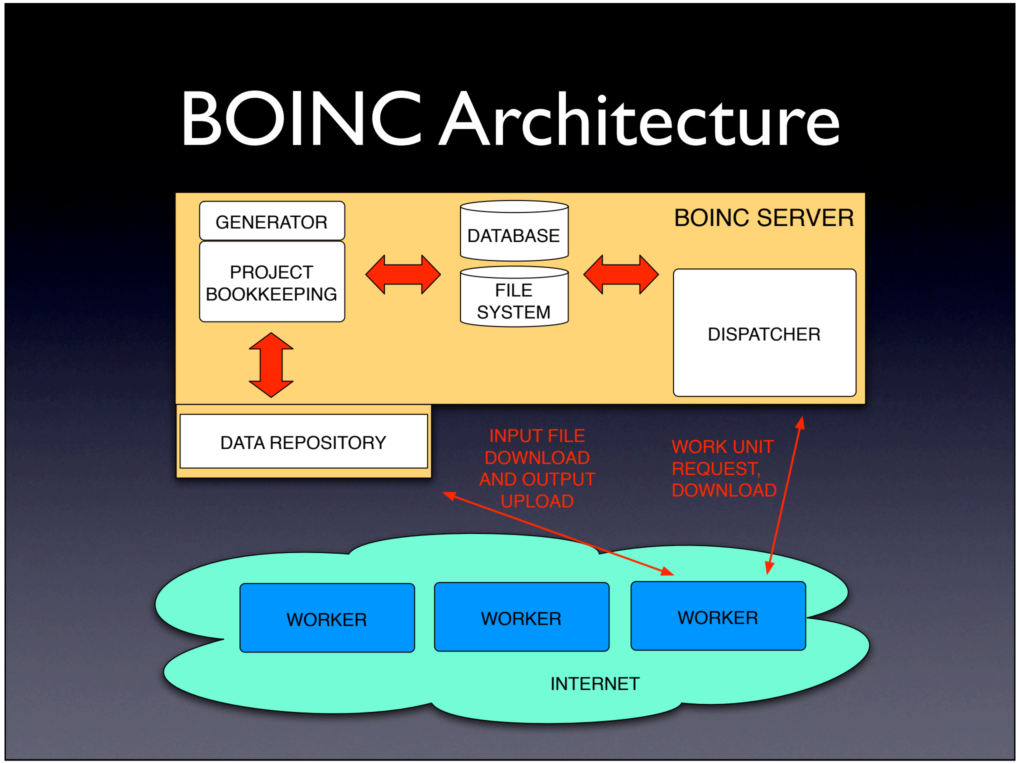
Outline

- 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

OOO install condor
globus: have to setup certificate authority, compile, permissions



data repository is just an http server that provides upload and download access to files

worker request includes description of the host and amount of work requested

input files (executable and input files) are downloaded from the data server
output files are uploaded to the data server

As described earlier, a workunit represents the inputs to a computation. The steps in creating a workunit are:

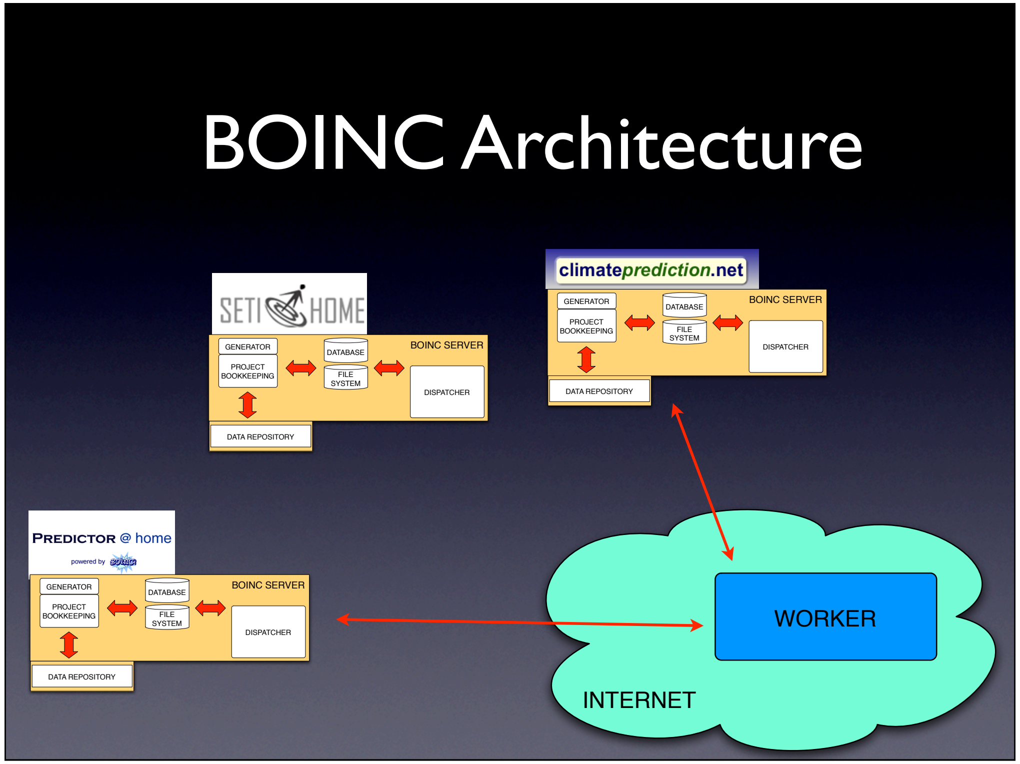
* Write XML 'template files' that describe the workunit and its corresponding results. Generally the same templates will be used for a large number work of workunits.
* Create the workunit's input file(s) and place them in the download directory.
* Invoke a BOINC function or script that creates a database record for the workunit.

Once this is done, BOINC takes over: it creates one or more results for the workunit, distributes them to client hosts, collects the output files, finds a canonical result, assimilates the canonical result, and deletes files.

is validator on different machine as the data repository

worker responsible for controlling when application can run

separation of data and accounting
how is data replicated between the data server and the scheduling server



use boinc for grid4all

- Scheduling done at the worker level (as their is no communication among the servers)

OOO replace einstein with LHC

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)

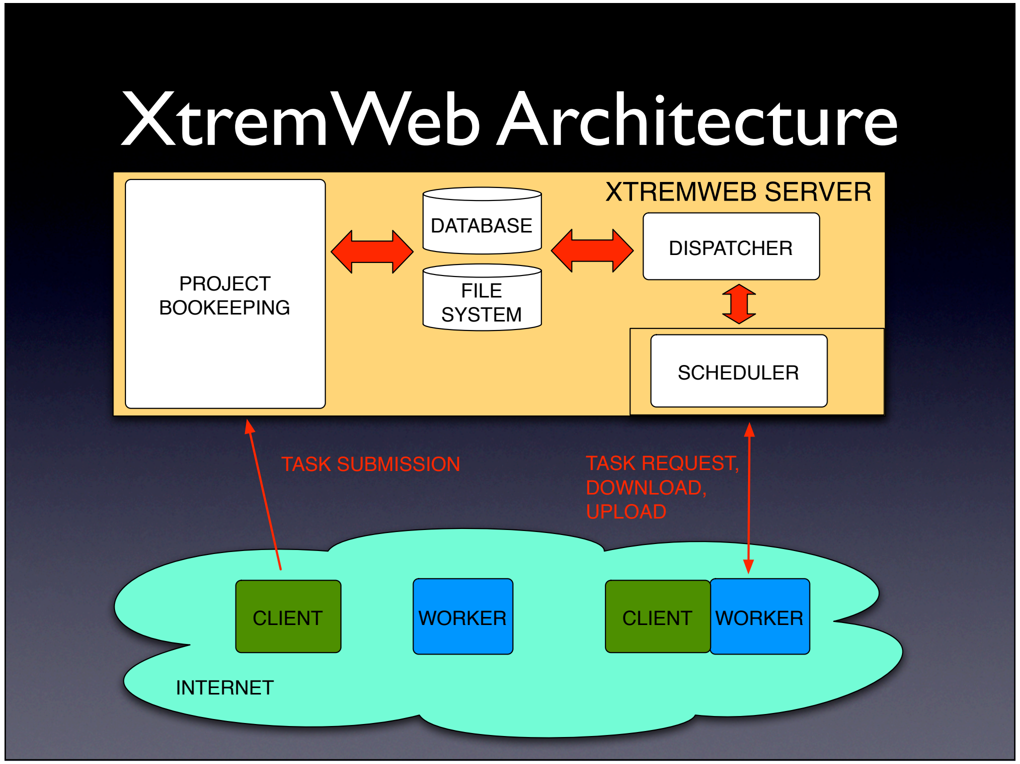
OOO scheduling approach
check number of parameters

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

BOINC assumed that the needs of users are asymmetric

true
where has xtremweb been used



difference between XW and BOINC, is that worker does not have choice about which application to participate in.

OOO multiple clients
coordination among schedulers: none
worker communication is done via xml-rpc (using http as the transport protocol) or java rmi
how does dispatcher choose tasks from applications: round robin

client, server, dispatcher, application executed determined by worker
what does user have to implement
relationship between file and database
replication allowed?
how to authenticate
differences bewteen boinc

Ensuring Collective Availability in Volatile Resource Pools via Forecasting

Artur Andrzejak Zuse-Institute Berlin (ZIB)

Derrick Kondo INRIA

David P. Anderson UC Berkeley

Motivation

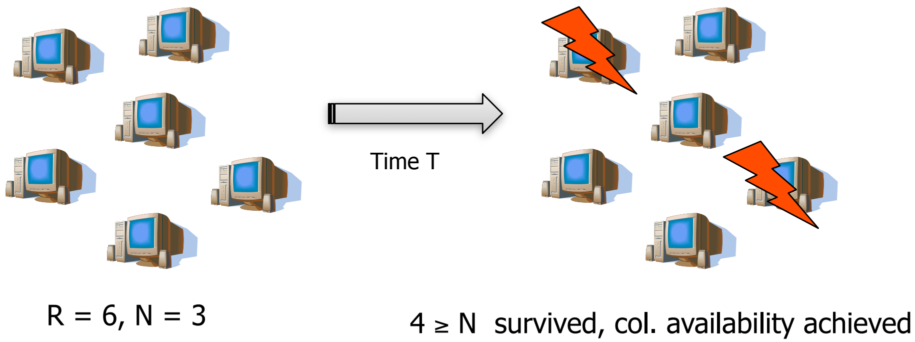
- Goal: can we deploy serious services / apps over unreliable resources?
- How unreliable?
  - mostly non-dedicated PC's (used for other purposes)
  - e.g. volunteer computing Grids such as SETI@home
  - no control over availability, frequent churn
- What are "serious" services / apps?
  - large scale service deployment
    - examples: Amazon's EC2, TeraGrid, EGEE
  - complex applications
    - examples: DAG/message-passing applications
  - high availability: around 99.999

Amazon Elastic Compute Cloud (Amazon EC2) is a web service that provides resizable compute capacity in the cloud. It is designed to make web-scale computing easier for developers.

Amazon EC2's simple web service interface allows you to obtain and configure capacity with minimal friction. It provides you with complete control of your computing resources and lets you run on Amazon's proven computing environment. Amazon EC2 reduces the time required to obtain and boot new server instances to minutes, allowing you to quickly scale capacity, both up and down, as your computing requirements change. Amazon EC2 changes the economics of computing by allowing you to pay only for capacity that you actually use. Amazon EC2 provides developers the tools to build failure resilient applications and isolate themselves from common failure scenarios.

How to do this?

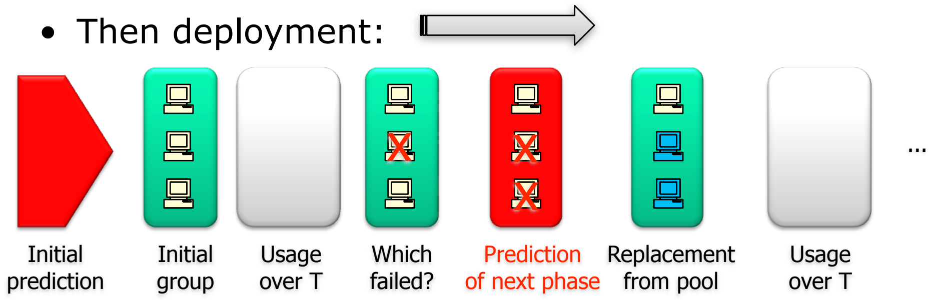
- Difficult to get (many) hosts with high avail
- Instead, we strive for collective availability:
  - def.: *guarantee that with high probability, in a group of R ≥ N hosts, at least N remain available over time T*



mean is .62%

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:



Availability Prediction

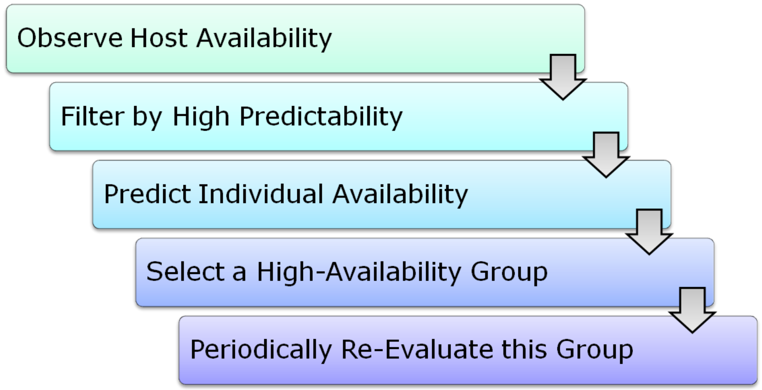
- We propose efficient and domain-adjusted predictions of availability for individual hosts
  - efficient:
    - fast pre-selection of predictable hosts
    - use simple and fast classification algorithm
  - domain-adjusted
    - analyze the factors of predictability and adjust our methods to them
-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

domain-adujusted: use right features like time of day

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:
  - Derrick Kondo, Artur Andrzejak, David P. Anderson: ***On Correlated Availability in Internet-Distributed Systems***, 9th IEEE/ACM International Conference on Grid Computing (Grid 2008), Tsukuba, Japan, September 29-October 1, 2008

Prediction Process



predictability is different from avialability. 1 hour

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

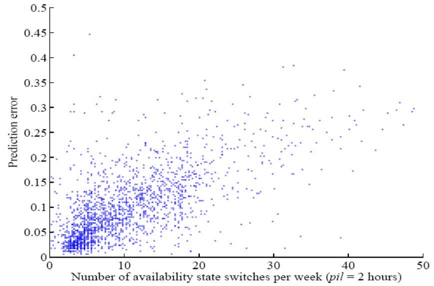
- We have tested, among others:
  - Average length of an uninterrupted availability segment
  - Size of the compressed availability trace
    - traces with predictable patterns are likely to compress better
  - Prediction error tested on a part of the training data (as a "control indicator")
  - Number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

OOO how to compress

And the winner is..

- Number of availability state changes per week: aveSwitches

Scatter plot
aveSwitches vs.
prediction error



Spearman's rank
correlation coefficient
indicator vs.
prediction error

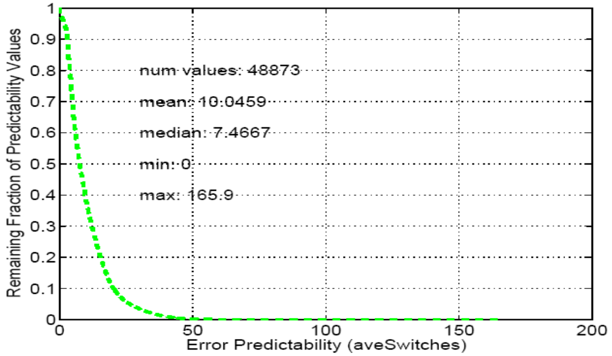
pil	**aveAva**	**aveAvaRun**	**aveNavaRun**	**aveSwitches**	**zipPred**	**modelPred**
1	-0.370	-0.594	0.085	0.707	-0.654	-0.724
2	-0.275	-0.486	-0.011	0.678	-0.632	-0.690
4	-0.119	-0.303	-0.119	0.548	-0.502	-0.640
8	0.194	0.056	-0.245	0.195	-0.127	-0.642
16	0.211	0.091	-0.185	0.057	0.062	-0.568

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 aveSwitches ≥ 7.47
  - high predictability, with 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

	$Attribute_1$	...	$Attribute_n$	**Output**
Example 1	[23,10]	...	[21,5]	0
Example k	[11,10]	...	[5,0]	1
Prediction	[1,7]	...	[13,7]	?

learn

predict

11-12

Time features include for each sample calendar information, such as hour in day, hour in week, day in week, day in month

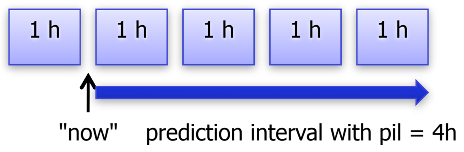
The hist features (for each sample) the sums of the recent k “history bits” for k = 2, 5, 10, 50, 100.

To help a classifier, we enrich the original 01 data with features from the folowing groups. The time features include for each sample calendar information such as hour in day, hour in week, day in week, day in month.   The hist features are (for each sample) the sums of the recent k “history bits”for k = 2, 5, 10, 20, 50, 100.

Predictive models are implemented using classifiers

Prediction

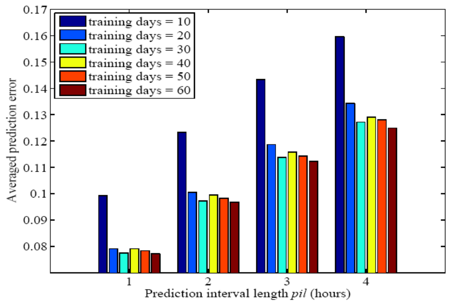
- We have used a simple and fast classifier
  - naive Bayes
- The classifier takes examples i.e. vectors of measured avail + preprocessed data over 30 days
- Predicts for each hour over two weeks
  - starting now, will the host be available in the next k hours
    - this is prediction interval length, pil

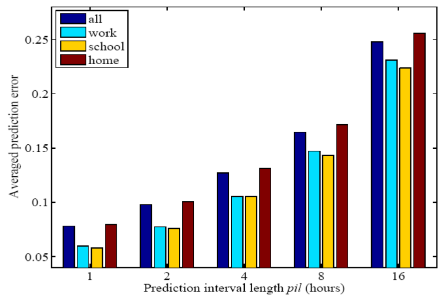


$p(A|B) = p(A)p(B|A) / p(B)$

What drives accuracy?

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





OOO filter out all hosts with avail < 1 day (filter out hosts just testing BOINC)

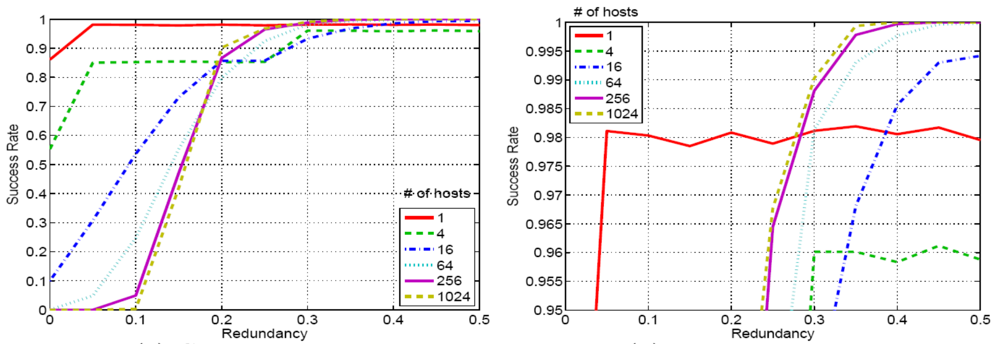
Simulation Approach

- For each host in the high-predictability group make prediction at t_0 for pil time, and select random R among those predicted as availiable
- 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 α, 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)

OOO number points
eric: OOO order by predictabiity
OOO remove 100% available hosts.
 (but need to consider host speed as well)

Necessary Redundancy

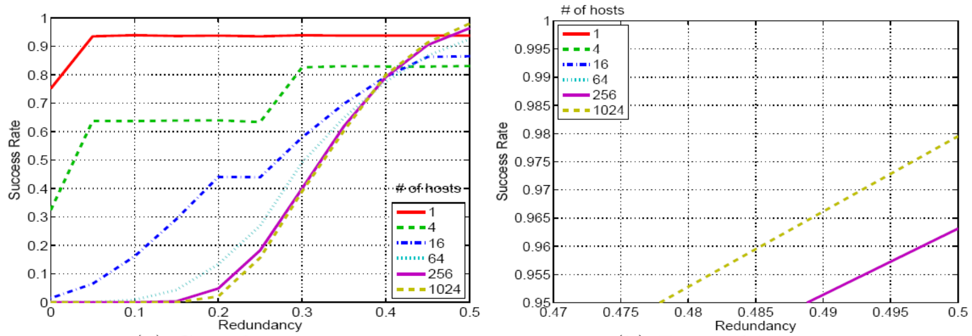
- High predictability group (pil=4)



(a) Complete range (b) Zoomed-in range

Necessary Redundancy

- Low predictability group (pil=4)



(a) Complete range (b) Zoomed-in range

Is this Redundancy too high?

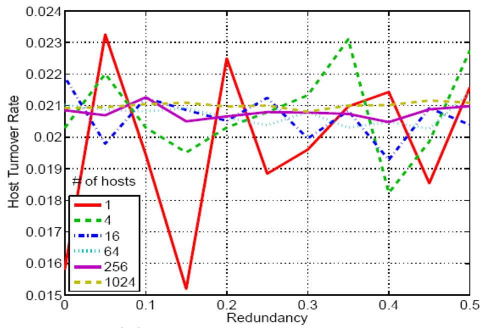
- In high predictability group, we have required redundancy of 35%
- However, we consider this dramatically low
  - In comparison, SETI@home has 200% redundancy (also used for result validation)
  - In terms of absolute savings, that equates to 165 TeraFLOPS saved in a 1 PetaFLOPS system (such as FOLDING@home) => 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

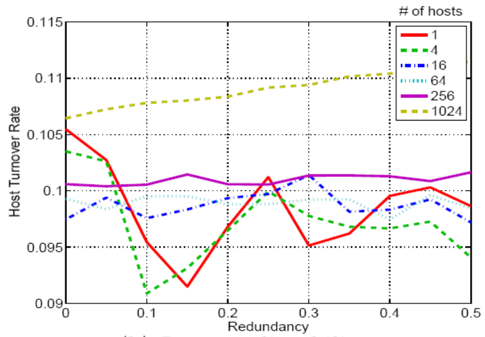
- We also evaluated the overhead due to host migration, service restart between slices of len T
- Threshold = a multiple of pil which describes the total time (many T's) of running an app / service
- Turnover rate TR:
  - let S be a set of hosts predicted to be available at t_0
  - for those we predict 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



(a) High predictability



(b) Low predictability

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:
  - 19th IFIP/IEEE Conference on Distributed Systems: Operations and Management (DSOM 2008) (part of Manweek 2008), Samos Island, Greece, September 22-26, 2008
- Pdf available on request, please send e mail (derrick.kondo [at] inria.fr)

Reverse Web-Link Graph

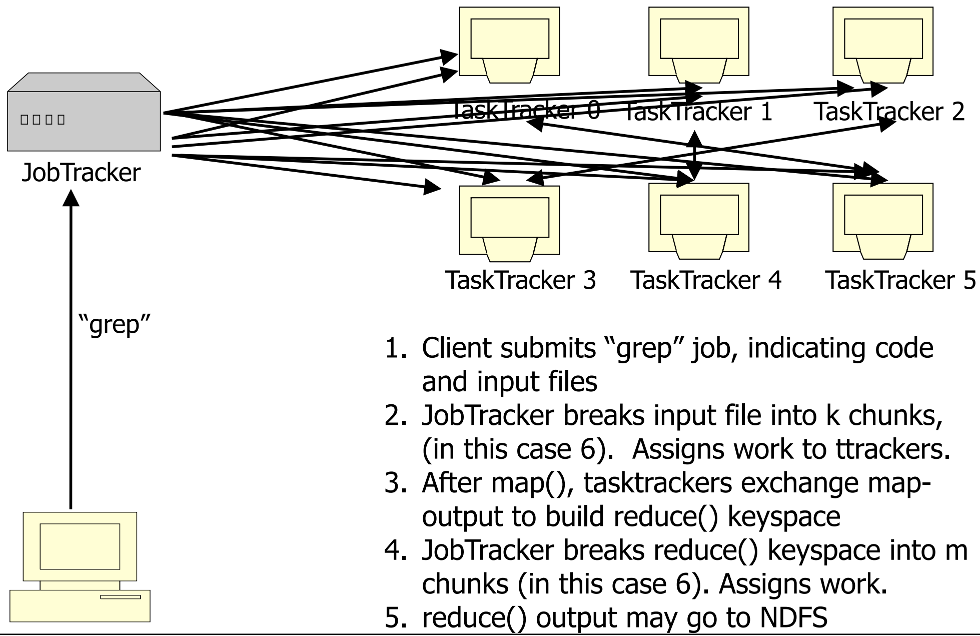
- Map
  - For each URL linking to target, …
  - Output <target, source> pairs
- Reduce
  - Concatenate list of all source URLs
  - Outputs: <target, **list** (source)> pairs

Inverted Index

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

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

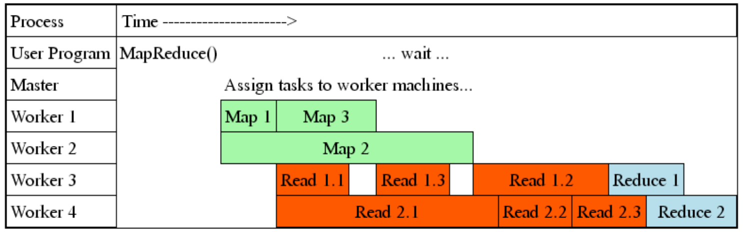
Job Processing



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
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

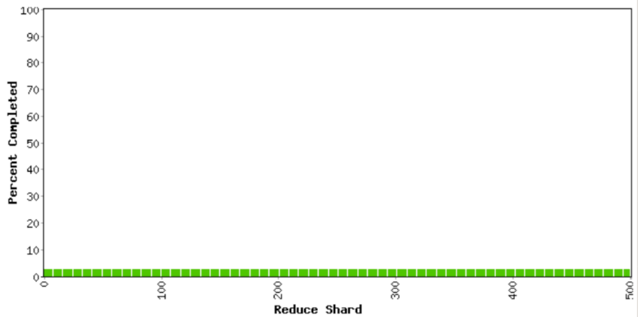


MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

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

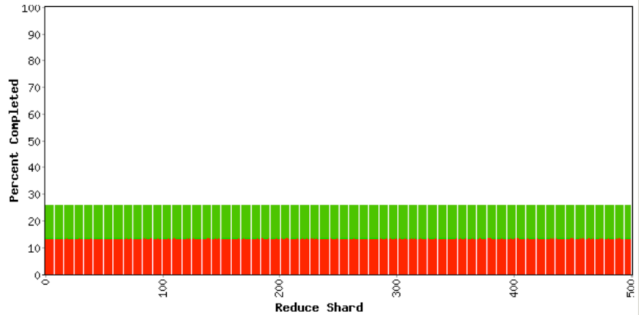
Variable	Minute
Mapped (MB/s)	72.5
Shuffle (MB/s)	0.0
Output (MB/s)	0.0
doc-index-hits	145825686
docs-indexed	506631
dups-in-index-merge	0
mr-operator-calls	508192
mr-operator-outputs	506631

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 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

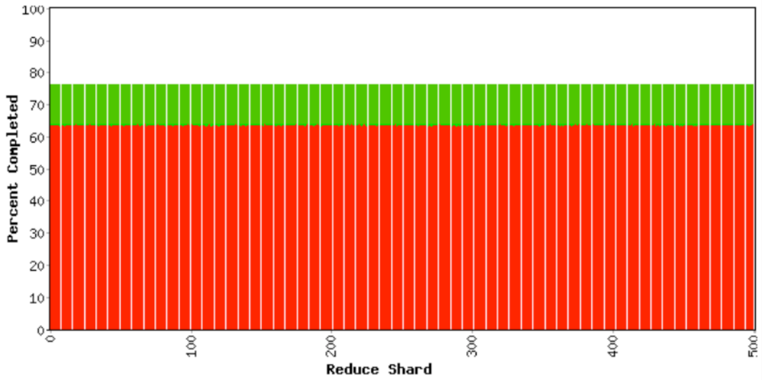
Variable	Minute
Mapped (MB/s)	704.4
Shuffle (MB/s)	371.9
Output (MB/s)	0.0
doc-index-hits	5000364228
docs-indexed	17300709
dups-in-index-merge	0
mr-operator-calls	17342493
mr-operator-outputs	17300709

MapReduce 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

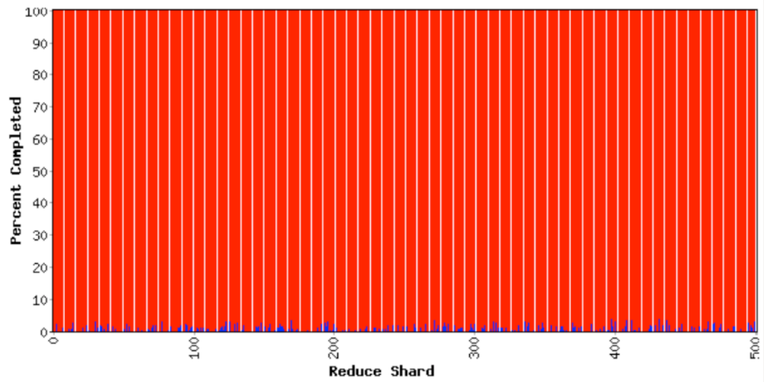
Variable	Minute
Mapped (MB/s)	706.5
Shuffle (MB/s)	419.2
Output (MB/s)	0.0
doc-index-hits	4982870667
docs-indexed	17229926
dups-in-index-merge	0
mr-operator-calls	17272056
mr-operator-outputs	17229926

MapReduce 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

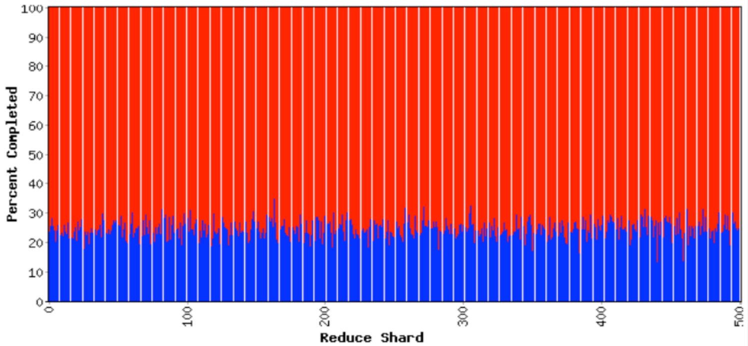
Variable	Minute	
Mapped (MB/s)	0.3	
Shuffle (MB/s)	0.5	
Output (MB/s)	45.7	
doc-index-hits	2313178	105
docs-indexed	7936	
dups-in-index-merge	0	
mr-merge-calls	1954105	
mr-merge-outputs	1954105	

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

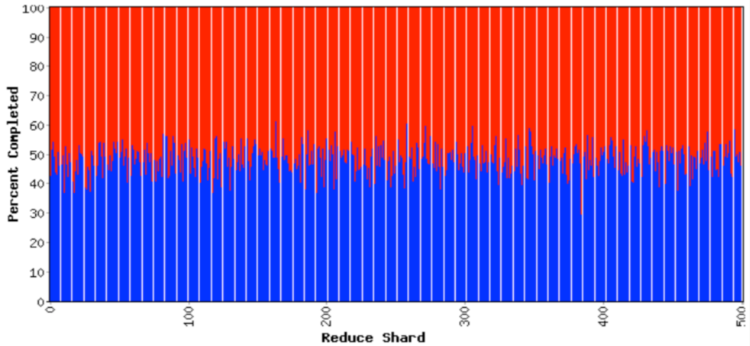
Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.1
Output (MB/s)	1238.8
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	51738599
mr-merge-outputs	51738599

MapReduce 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

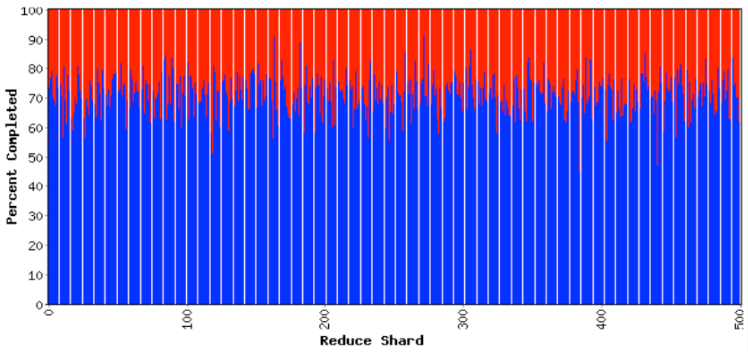
Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	1225.1
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	51842100
mr-merge-outputs	51842100

MapReduce 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

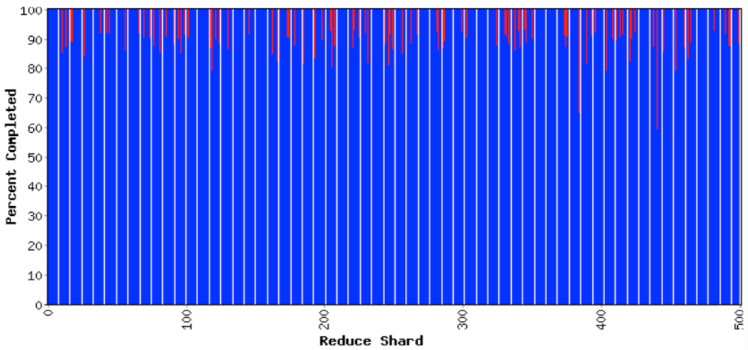
Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1222.0	
doc-index-hits	0	1(
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	51640600	
mr-merge-outputs	51640600	

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

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

1707 workers; 1 deaths

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



Counters

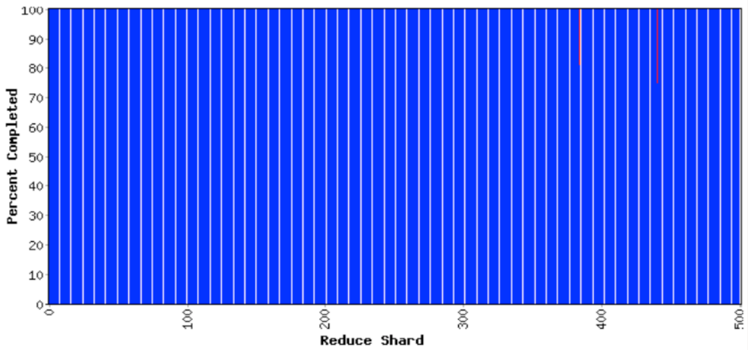
Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	849.5
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	35083350
mr-merge-outputs	35083350

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

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

1707 workers; 1 deaths

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



Counters

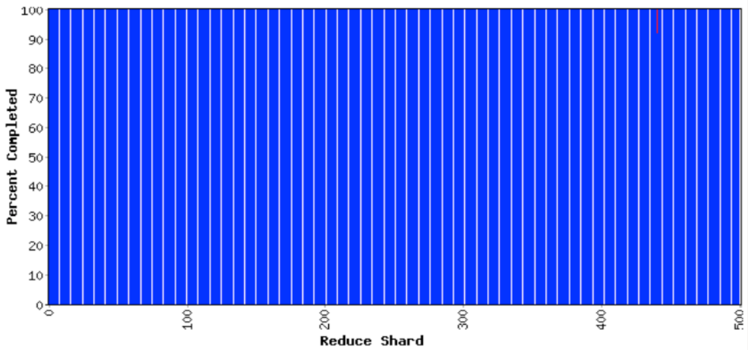
Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	9.4	
doc-index-hits	0	105[illegible]
docs-indexed	0	[illegible]
dups-in-index-merge	0	
mr-merge-calls	394792	[illegible]
mr-merge-outputs	394792	[illegible]

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	

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)

- Assumptions defined by Globus

Steps

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