Distributed Computing: MapReduce and Beyond!

Google, Inc.
Jan. 14, 2008

Distributed Computing

Distributed ...

- programs: run on two or more networked computers.
- algorithms: distributed programs that terminate.
- systems: distributed programs that run indefinitely, usually in order to provide one or more services.

- Example distributed programs:

SETI@home, DNS, BitTorrent, Google, ...

Why Distributed Computing is Important

- Users are distributed.
- The information is distributed.
- Can be more reliable.
- Can be faster.
- Can be cheaper ($30 million Cray versus 100 $1000 PC's).

Why Distributed Computing is Hard

- Computers crash.
- Network links crash.
- Talking is slow (ranges from 56 kbit/s modem to 10 Gbit/s ethernet, but even ethernet has ~300 microsecond latency, during which time your 2 Ghz PC can do 600,000 cycles).
- Bandwidth is finite.
- There's no global state or clock.
- Internet scale: the computers and network are heterogeneous, untrustworthy, and subject to change at any time.

"*A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.*"
-- Leslie Lamport

Scales of Distributed Programming

Name	Example	# Processors	Network	Bandwidth	Issues
Multicore	Intel Core 2	4	Memory	10.66 GB/s	Thread safety
Multiprocessor	IBM p690	32	Memory, Bus	20 GB/s	Bus saturation, cache coherency
Cluster, Tight	Blue Gene/L	66560	LAN	0.3 GB/s	Component failure, async.
Cluster, Loose	Our cluster	75	LAN	1-10 Gb/s	Small heterogeneity
Grid	BOINC	10^9+	Internet	variable	Trust, change, big heterogeneity

Three Common Distributed Architectures

• Hope: have N computers do separate pieces of work. Speed-up < N. Probability of failure = 1 – (1-p)^N ≈ Np. (p = probability of individual crash).

• Replication: have N computers do the same thing. Speed-up < 1. Probability of failure = p^N.

• Master-servant: have 1 computer hand out pieces of work to N-1 servants, and re-hand out pieces of work if servants fail. Speed-up < N-1. Probability of failure ≈ p.

It would be nice to be able to replicate the master ...
on Wednesday, we'll talk about a robust way of doing so.

Example Distributed System: Google File System

- GFS is a distributed file system written at Google for Google's needs (lots of data, lots of cheap computers, need for speed).
- We use it to store the data from our web crawl, book search, GMail, ...
- It's a good example of a real world distributed system.
- Hadoop's file system is based on it.

More details about GFS can be found in “The Google File System” paper at http://labs.google.com/papers/gfs-sosp2003.pdf .

What a Distributed File System Does

1) Usual file system stuff: create, read, move & find files.

2) Allow distributed access to files.

3) Files are stored distributedly.

If you just do #1 and #2, you are a network file system.

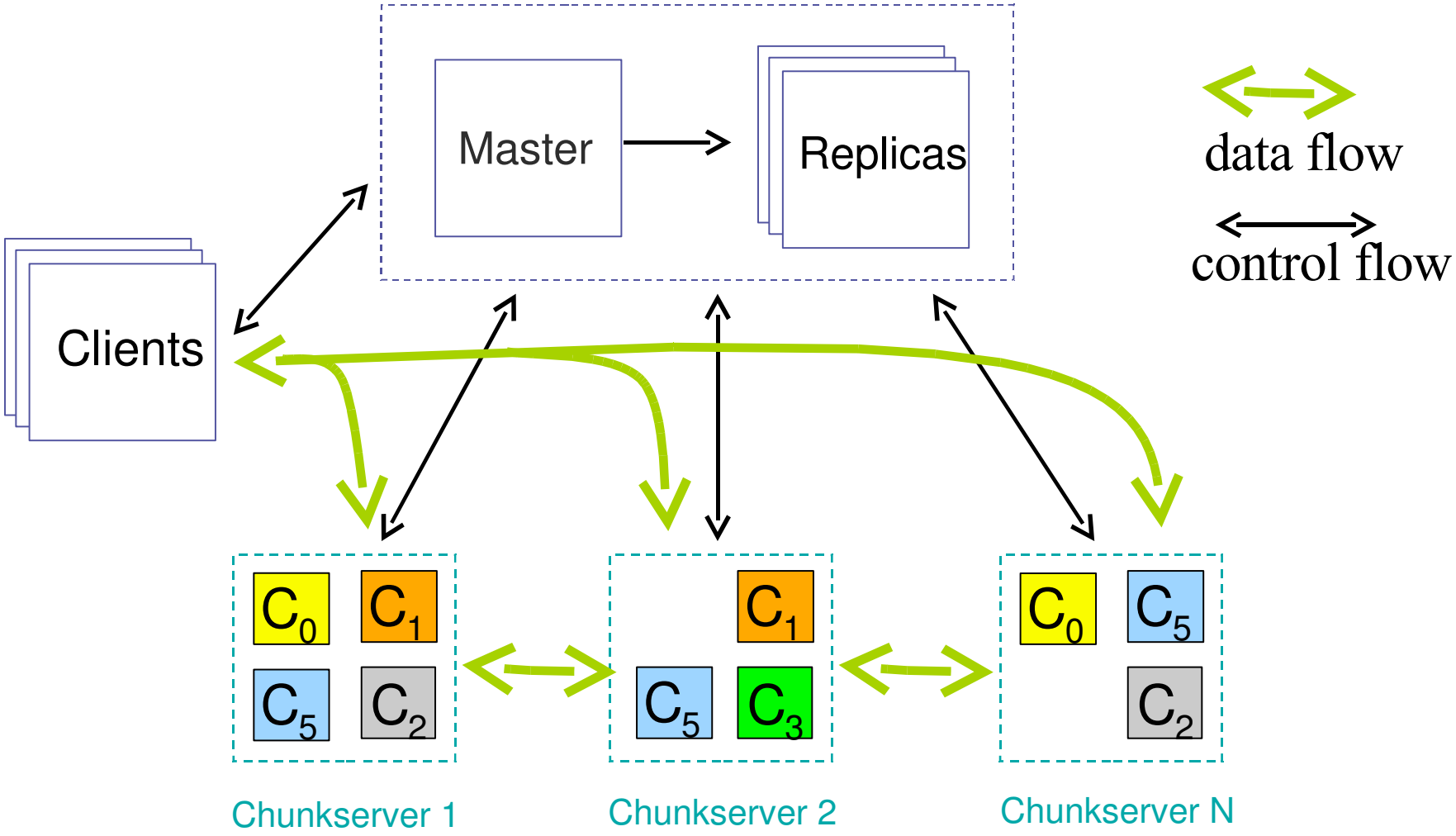
To do #3, it's a good idea to also provide fault tolerance.

How GFS is Different From Other Distributed FS's

Based on Google's workload, GFS assumes

- High failure rate
- Huge files (optimized for GB+ files)
- Almost all writes are appends
- Reads are either small and random OR big and streaming
- There's no need to implement POSIX (no symbolic links, etc.)
- Throughput is more important than latency

GFS Architecture



Google™

GFS Architecture: Chunks

- Files are divided into 64 MB chunks (last chunk of a file may be smaller).
- Each chunk is identified by an unique 64-bit id.
- Chunks are stored as regular files on local disks.
- By default, each chunk is stored **thrice**, preferably on more than one rack.
- To protect data integrity, each 64 KB block gets a 32 bit checksum that is checked on all reads.
- When idle, a chunkserver scans inactive chunks for corruption.

GFS Architecture: Master

- Stores all metadata (namespace, access control).
- Stores (file -> chunks) and (chunk -> location) mappings.
- All of the above is stored in memory for speed.
- Clients get chunk locations for a file from the master, and then talk directly to the chunkservers for the data.
- Master is also in charge of chunk management: migrating, replicating, garbage collecting and leasing chunks.
- Advantage of single master: simplicity.
- Disadvantages of single master:
  - Metadata operations are bottlenecked.
  - Maximum # of files limited by master's memory.

GFS: Life of a Read

1) Client program asks for 1 Gb of file “A” starting at the 200 millionth byte.
2) Client GFS library asks master for chunks 3, ... 16387 of file “A”.
3) Master responds with all of the locations of chunks 2, ... 20000 of file “A”.
4) Client caches all of these locations (with their cache time-outs).
5) Client reads chunk 2 from the closest location.
6) Client reads chunk 3 from the closest location.
7) ...

GFS: Life of a Write

1) Client gets locations of chunk replicas as before.

2) For each chunk, client sends the write data to nearest replica.

3) This replica sends the data to the nearest replica to it that has not yet received the data.

4) When all of the replicas have received the data, then it is safe for them to actually write it.

Tricky details:

- Master hands out a short term (~1 minute) lease for a particular replica to be the primary one.
- This primary replica assigns a serial number to each mutation so that every replica performs the mutations in the same order.

GFS: Atomic Appends and Snapshots

- GFS provides two non-traditional but incredibly useful operations.
- Atomic append: normal writes guarantee all replicas will be the same, but aren't atomic when used to append. Atomic append guarantees the entire append is atomic, but replicas may differ.
- Implementation:

  1) Pad all replicas if append wouldn't fit inside current last chunk.

  2) Append to primary replica.

  3) Try appending to all other replicas at the same offset.

  4) If any fail, try over.

- Snapshot: almost instantaneous copy of a file or directory tree. Implemented by expiring all leases on the effected chunks and using copy-on-write.

GFS: Life of a Chunk

Initial chunk placement balances several factors:

- Want to use under-utilized chunkservers.
- Don't want to overload a chunkserver with lots of new chunks.
- Want to spread a chunk's replicas over more than one rack.

A chunk is re-replicated if it has too few replicas (because a chunkserver fails, or a checksum detects corruption, ...).

Master also periodically rebalances chunk replicas. (New chunkservers will get used even if no new chunks are being created.)

GFS: Master Failure

- The Master stores its state via periodic checkpoints and a mutation log.
- Both are replicated.
- Master election and notification is implemented using an external lock server (“Stubby”; we'll talk about it's fundamental algorithm on Wed.).
- New master restores state from checkpoint and log (exception: chunk locations are determined by asking the chunkservers).
- Takes minutes (users would prefer seconds).
- Master shadows are also used for non-mutating operations.

Differences between GFS & Hadoop Distributed FS

HDFS does not yet implement:

- more than one simultaneous writer
- writes other than appends
- automatic master failover
- snapshots
- optimized chunk replica placement
- data rebalancing

HDFS Interface

Shell:
bin/hadoop dfs -{command}

For example,

bin/hadoop dfs -mkdir /foodir

bin/hadoop -cat /foodir/myfile.txt

Java:

org.apache.hadoop.dfs.DistributedFilesystem supplies

create, open, exists, rename, delete, mkdirs, listPaths (ls),

getFileStatus (stat), set & getWorkingDirectory, etc.

Next Time

We'll talk about all the things Mapreduce saves you from:

- Distributed infrastructures for communication
- Deadlock
- Master election with Paxos.