Distributed systems case studies

Dr. Richard E. Turner (ret26@cam.ac.uk)

February 23, 2015
Distributed systems architecture underpinning Google

- **Background about services**
  - Brief review to motivate demands on physical model, comms and services

- **Physical model and architecture**
  - Hardware setup

- **Comms**
  - How distributed systems are made to look like single systems

- **Services**
  - Data structures
Distributed systems architecture underpinning Google

- Mission statement “to organise the world’s information and make it universally accessible and useful”

- Born out of an internet search research project at Stanford, now diversified into cloud computing

- **Cloud computing** = a set of internet based application, storage and computing services sufficient to support most users’ needs thus allowing them to dispense with local data-storage and application software

Larry Page
(co-founder, CEO)

Eric Schmidt
(Executive Chairman)

Sergey Brin
(co-founder)
Scalability

• initial production system in 1998 (Larry Page and Sergey Brin run company from a garage) to 5 billion searches per day – 50,000 searches per second

• main search engine has never experienced an outage

• on ave. users receive query results in .2seconds

• How is this possible?
Google search engine

- **Crawling**
  - Googlebot carries out deep searching: recursively reads a webpage, harvests links, schedules further crawls on the harvested links
  - visits pages in proportion to how much they’ve changed historically
  - used to be every two weeks, now continuous
  - supported by Bigtable (infrastructure service for huge tables)

- **Indexing**
  - inverted index produced mapping words onto positions they occur in documents
  - stores meta data like font size and position
  - sorted (indexed) to support efficient queries for words against locations
  - tends to narrow down candidate webpages from billions to tens of thousands on average
  - index of links (which webpages link to a given site) also maintained (c.f. pagerank)
Google search engine

- Ranking
  - need relative importance of webpages
  - Page rank is a key reason for the success of Google
  - inspired by citation indices:
  - webpage important if linked to by many other webpages
  - extends this idea: webpage more important if important sites link to it
  - also takes into account proximity of key words, how near to the start of the page they are, font size/capitalisation etc.
Cloud services

- **software as a service**
  - Google apps includes Gmail, Google Docs, Google Calendar etc.
  - encourages an open approach to innovation within the company, but this places particular constraints and demands on the infrastructure

- **platform as a service**
  - offer distributed system application programming interface (API): protocol used as an interface by software components to communicate with each other
  - make available parts of the cloud architecture that is used internally to support applications and services
  - e.g. organisations can use this to develop their own Google Apps.
Architecture and design philosophy

- Physical architecture: First server rack
  - scavenged 10 4GB hard drives
  - lego storage rack
Architecture and design philosophy

• **Physical architecture**
  – machines organised into racks of 40-80 + ethernet switch
  – machines: commodity PCs
  – cost effective environment for distributed storage and computation
Architecture and design philosophy

- purchase based on performance per dollar (usually $1000 per unit)
- 2 terabytes disc space, 16Gigs RAM, stripped down linux kernel

⇒ part of the infrastructure WILL FAIL
Architecture and design philosophy

- software most common failure: reboot 20 machines a day manually (!)
- 2-3% have hardware failures per annum (most faulty discs or RAM)
- most failures software $\Rightarrow$ no need to purchase more expensive machines
Data-centres

- racks organised into clusters containing about 30 racks (two high bandwidth switches providing connectivity to the outside world, redundant links)
Data-centres

- racks organised into clusters containing about 30 racks (two high bandwidth switches providing connectivity to the outside world, redundant links)
- clusters housed in data-centres

**Americas**
- Berkeley County, South Carolina
- Council Bluffs, Iowa
- Douglas County, Georgia
- Quilicura, Chile
- Mayes County, Oklahoma
- Lenoir, North Carolina
- The Dalles, Oregon

**Asia**
- Changhua County, Taiwan
- Singapore

**Europe**
- Hamina, Finland
- St Ghislain, Belgium
- Dublin, Ireland
- Eemshaven, Netherlands
Architecture requirements

- **scalability**
  - dealing with more data & more queries, seeking better results

- **reliability**
  - 99.9% service agreement to Google Apps paying customers
  - 1/9/2009 Gmail down 100min (maintenance: cascade overloaded servers)
  - easy to mask websearch failures (some of the results not returned)

  ⇒ need to detect and mask inevitable failures

- **performance**
  - 0.2s target for returning web search queries - depends on whole pipeline

- **openness**
  - infrastructure needs to be extensible and therefore open to provide support for new applications
Design philosophy

• **simplicity**: “software should do one thing well”
  – avoid feature rich designs
  – avoid monolithic code, write small functions that are simple to test and reuse

• **performance**: “every millisecond counts”
  – estimate speed using back of the envelope calculations
  – plan your programme with a pen and paper before coding

• **testing**: “if it aint broke, you’re not trying hard enough”
  – unit and integration tests
Middleware: software glue

- **middleware** = software providing services to software applications, beyond that provided by the operating system
  - software layer between the O/S and applications on each side of a distributed computing system in a network
  - e.g. the dash in “client-server”

- makes it easier for software developers to perform
  - communication
  - input/output

- developers can then focus on the specific purpose of their application.
Remote procedure calls

Goal: make programming in distributed systems look like conventional programming
Remote procedure calls

possibly different platforms, languages, operating systems
cannot pass pointers around (address spaces on two machines different)
call by reference is out, cannot have direct access to variables
Remote Procedure Call: procedure on remote machines called as if they were in the local address space. It abstracts away details of creating/passing/reading messages.
Remote procedure calls

marshalling: piece of code called the stub packs the parameters into a message (converts into processor/language/address-space-neural format)
Remote procedure calls

unmarshalling: at the server-end the server-stub unpacks parameters from the messages, calls the server procedure
Remote procedure calls

sends the results back by a reverse of the above steps
• **Remote method invocation**: object oriented version of a RPI
  - natural: *interfaces* abstract away how the service is implemented (language, OS, platform etc.)

• **Protocol buffers**
  - language for specifying data-formats that supports RPCs across the network
  - compiler produces abstract interface and a stub
  - 3–10 times smaller than XML, 10–100 times faster, but Google’s infrastructure more constrained
Google File System

- Access to **unstructured data** (e.g. holds data for performing indexing)

- Lots of commodity hardware \(\Rightarrow\) **there will be failures**

- relatively small number of massive files (each 100s megabytes in 2003)

- **access**: normally sequential reads (for performing indexing) and sequential writes (updating the stored webpage data)

- **concurrency**: concurrent reads and appends very common

- Conventional interface: create, delete, open, close, \{read, write\} (take offset within file) specialised operations: snapshot (backup), record append (write to the end)
Google File System

one master
manages metadata (namespace),
access control,
mapping of files to chunks

GFS master
file metadata

100s of chunkservers
5 replicas are made of each chunk

GFS chunkserver
data chunks

GFS chunkserver
data chunks

write ahead logs support recovery

data chunked into 64Mb pieces - reduces amount of metadata so that master can keep this in main memory
Client requests read or write of file:
GFS client library converts request into filename-chunk pair
client requests a mutation: sent via protocol buffers to GFS master
Master grants chunk lease to a replica (the primary) also returns the identity of the primary and secondaries to client
client sends data to all replicas - stored in buffer (not written yet)
primary produces serial order for all actions pending on chunk
global (serial) ordering produced by chunk-leases & primary order
once data has been received, primary applies changes
tells secondaries to apply the same changes
if there's a failure, the primary retries mutation on secondaries
Google File System

Summary:
Data and control flows are separated, Master gets out of way ASAP
Master sees whole system - makes optimal management decisions
Chubby

- abstraction based upon a file system: **every data object is a file**
  /ls/chubby_cell/directory_name/.../file_name

- **file system** for storing small files reliably
  
  chubby_cell refers to a particular instance of the Chubby system

- **coarse grained distributed locks** to synchronise distributed activities
  
  /ls refers to the lock service

- **advisory locking service**
  - no central lock manager that enforces the locks
  - programmer must do this and check for conflicts (locking locks out other lockers, but that’s it)

- supports election of primary and secondary replicas
Chubby in elections

- all candidate primaries try and obtain a lock
- one succeeds - the primary
- primary records victory by writing identity to associated file
- other processes can determine the identity of the primary by reading this data
Bigtable

store vast volumes of structured data petabyte range across “hundreds or thousands of machines”
simpler than a relational database
for indexing: webpage metadata stored using URL key

google earth: images stored by geographical segments

google analytics: user activity by user sessions
Bigtable

- **Row**: unique key
- **Column**: time stamp

Setup to perform garbage collection after some time-lag.

**For indexing**: webpage metadata stored using URL key.
**Google Earth**: images stored by geographical segments.
**Google Analytics**: user activity by user sessions.
Bigtable

- row: unique key
- column
- timestamp

setup to perform garbage collection after some time-lag

for indexing: webpage metadata stored using URL key

google earth: images stored by geographical segments

google analytics: user activity by user sessions
Bigtable

- The time stamp setup to perform garbage collection after some time-lag.
- Unique key column.
- Tablet 1 (~200Mb)
- Tablet 2
- Tablet 3

- Sparse

For indexing: Webpage metadata stored using URL key.

Google Earth: Images stored by geographical segments.

Google Analytics: User activity by user sessions.
Bigtable

**cluster**: collection of tables
Summary:
Data & control flows are separated, Master not involved with access
Master sees whole system - makes optimal management decisions
Summary:
Data & control flows are separated, Master not involved with access
Master sees whole system - makes optimal management decisions
Summary:
Data & control flows are separated, Master not involved with access
Master sees whole system - makes optimal management decisions
Summary

- three relatively simple, complementary data storage services
- optimised for different applications
- used in combination as building blocks to produce sophisticated solutions