A Behind-the-Scenes Tour

Jeff Dean
Google Fellow
jeff@google.com

User's View of Google

Organizing the world's information and making it universally accessible and useful

A Computer Scientist's View of Google

Problems span a wide range of areas:

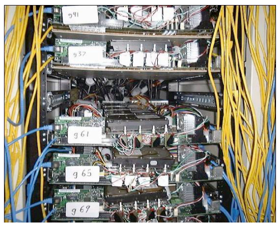
Product design	…and much, much more!
User interfaces	
Machine learning, Statistics, Information retrieval, AI	
Data structures, Algorithms	
Compilers, Programming languages	
Networking, Distributed systems, Fault tolerance	
Hardware, Mechanical engineering	

Hardware Design Philosophy

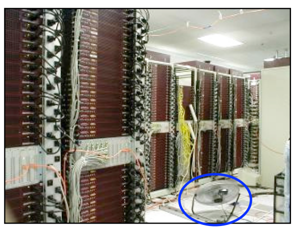
- Prefer low-end server/PC-class designs
  - Build lots of them!
- Why?
  - Single machine performance is not interesting
    - Even smaller problems are too large for any single system
    - Large problems have lots of available parallelism

“Google” Circa 1997 (google.stanford.edu)

Google (circa 1999)



Google Data Center (Circa 2000)

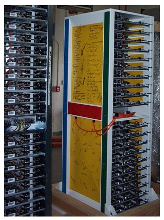


Google (new data center 2001)

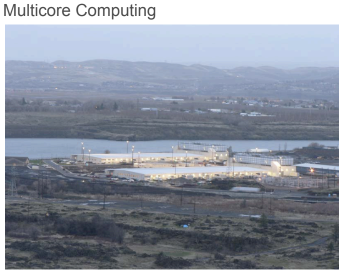
Google Data Center (3 days later)

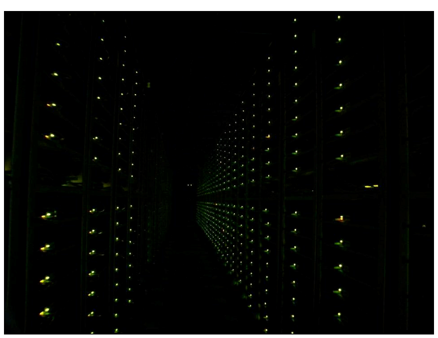
Current Design

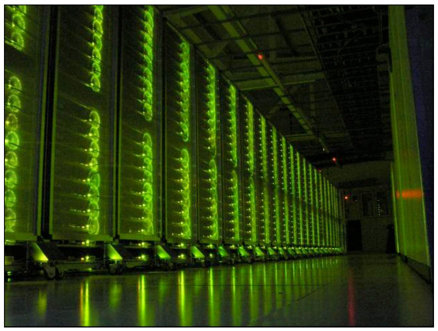
- In-house rack design
- PC-class motherboards
- Low-end storage and networking hardware
- Linux
- + in-house software



Multicore Computing







The Joys of Real Hardware

Typical first year for a new cluster:

~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)
~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)
~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hours)
~1 network rewiring (rolling ~5% of machines down over 2-day span)
~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
~5 racks go wonky (40-80 machines see 50% packetloss)
~8 network maintenances (4 might cause ~30-minute random connectivity losses)
~12 router reloads (takes out DNS and external vips for a couple minutes)
~3 router failures (have to immediately pull traffic for an hour)
~dozens of minor 30-second blips for dns
~1000 individual machine failures, ~thousands of hard drive failures
slow disks, bad memory, misconfigured machines, flaky machines, etc.

- Long-haul networking breaks for unusual reasons, too:
  - Wild dogs, dead horses, thieves, blasphemy, drunken hunters and sharks

Implications of our Computing Environment

Stuff Breaks

- If you have one server, it may stay up three years (1,000 days)
- If you have 10,000 servers, expect to lose ten a day

"Ultra-reliable" hardware doesn't really help

- At large scales, super-fancy reliable hardware still fails, albeit less often
  - software still needs to be fault-tolerant
  - commodity machines without fancy hardware give better perf/$

Reliability has to come from the software

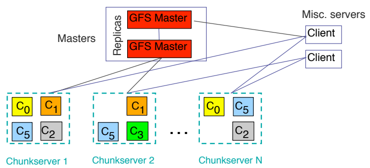
How can we make it easy to write distributed programs?

Google

Rest of Talk

- Overview of Infrastructure
  - GFS, MapReduce, BigTable
- A peek at machine translation
- Some fun with interesting data
- General software engineering style/philosophy

GFS Design



- Master manages metadata
- Data transfers happen directly between clients/ chunkservers
- Files broken into chunks (typically 64 MB)

GFS Usage @ Google

- 200+ clusters
- Many clusters of 1000s of machines
- Pools of 1000s of clients
- 4+ PB Filesystems
- 40 GB/s read/write load
- (in the presence of frequent HW failures)

MapReduce

- A simple programming model that applies to many large-scale computing problems
- Hide messy details in MapReduce runtime library:
  - automatic parallelization
  - load balancing
  - network and disk transfer optimizations
  - handling of machine failures
  - robustness
  - **improvements to core library benefit all users of library!**

Typical problem solved by MapReduce

- Read a lot of data
- Map: extract something you care about from each record
- Shuffle and Sort
- Reduce: aggregate, summarize, filter, or transform
- Write the results

Outline stays the same,
map and reduce change to fit the problem

Processing Large Datasets

Geographic Data — Index Files — Data Center

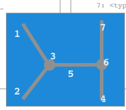
Transforming data

Input

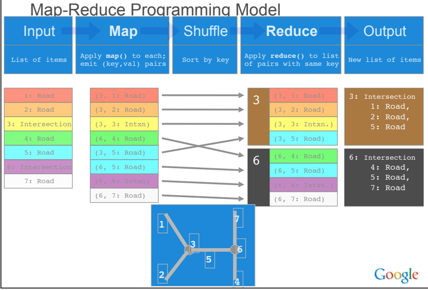
```
Feature List
1: <type=Road>, <intersections=(3)>, <geom>, …
2: <type=Road>, <intersections=(3)>, <geom>, …
3: <type=Intersection>, <roads=(1,2,5)>, …
4: <type=Road>, <intersections=(6)>, <geom>,
5: <type=Road>, <intersections=(3,6)>, <geom>, …
6: <type=Intersection>, <roads=(5,6,7)>, …
7: <type=Road>, <intersections=(6)>, <geom>, …
8: <type=Border>, <name>, <geom>, …
.
.
.
```

Output

```
Intersection List
3: <type=Intersection>, <roads=(
    1: <type=Road>, <geom>, <name>, …
    2: <type=Road>, <geom>, <name>, …
    5: <type=Road>, <geom>, <name>, …)>, …
6: <type=Intersection>, <roads=(
    4: <type=Road>, <geom>, <name>, … >
    5: <type=Road>, <geom>, <name>, … >
    7: <type=Road>, <geom>, <name>, …)>, …
.
.
.
```

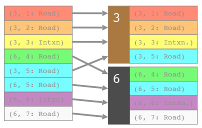


Map-Reduce Programming Model



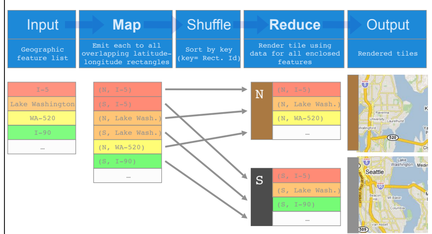
Code Example

```
class IntersectionAssemblerMapper : public Mapper {
  virtual void Map(MapInput* input) {
    GeoFeature feature;
    feature.FromMapInput(input);
    if (feature.type()==FEATURE_INTERSECTION) {
      Emit(feature.id(), input);
    } else if (feature.type() == FEATURE_ROAD) {
      Emit(feature.intersection_id(0), input);
      Emit(feature.intersection_id(1), input);
    }
  }
};
REGISTER_MAPPER(IntersectionAssemblerMapper);
```




```
class IntersectionAssemblerReducer:public Reducer {
  virtual void Reduce(ReduceInput* input) {
    GeoFeature feature;
    GraphIntersection intersection;
    intersection.id = input->key();
    while(!input->done()) {
      feature.FromMapInput(input->value());
      if (feature.type()==FEATURE_INTERSECTION)
        intersection.SetIntersection(feature);
      else
        intersection.AddRoadFeature(feature);
      input->next();
    }
    Emit(intersection);
  }
};
REGISTER_REDUCER(IntersectionAssemblerReducer);
```

Rendering Map Tiles



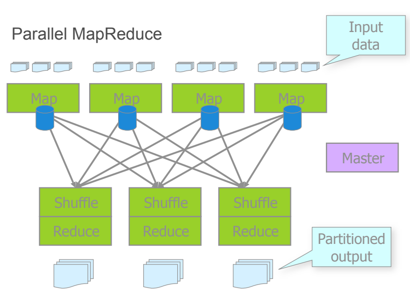
Widely applicable at Google

- Implemented as a C++ library linked to user programs
- Can read and write many different data types

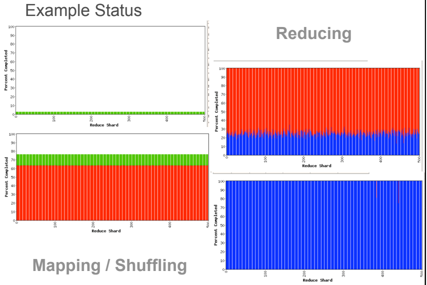
Example uses:

distributed grep distributed sort term-vector per host document clustering machine learning ...	web access log stats web link-graph reversal inverted index construction statistical machine translation ...

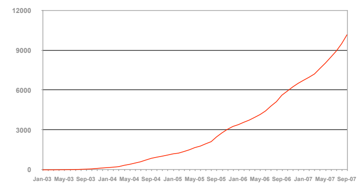
Parallel MapReduce



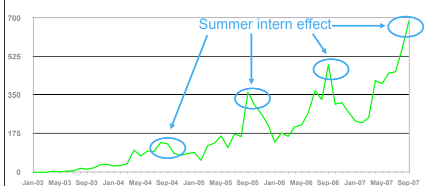
Example Status



MapReduce Programs in Google's Source Tree



New MapReduce Programs Per Month



Usage Statistics Over Time

	Aug, '04	Mar, '05	Mar, '06	Sep, '07
Number of jobs	29K	72K	171K	2,217K
Average completion time (secs)	634	934	874	395
Machine years used	217	981	2,002	11,081
Input data read (TB)	3,288	12,571	52,254	403,152
Intermediate data (TB)	758	2,756	6,743	34,774
Output data written (TB)	193	941	2,970	14,018
Average worker machines	157	232	268	394

BigTable: Motivation

- Lots of (semi-)structured data at Google
  - URLs:
    - Contents, crawl metadata, links, anchors, pagerank, …
  - Per-user data:
    - User preference settings, recent queries/search results, …
  - Geographic locations:
    - Physical entities (shops, restaurants, etc.), roads, satellite image data, user annotations, …
- Scale is large
  - billions of URLs, many versions/page (~20K/version)
  - Hundreds of millions of users, thousands of q/sec
  - 100TB+ of satellite image data

Google

Why not just use commercial DB?

- Scale is too large for most commercial databases
- Even if it weren't, cost would be very high
  - Building internally means system can be applied across many projects for low incremental cost
- Low-level storage optimizations help performance significantly
  - Much harder to do when running on top of a database layer

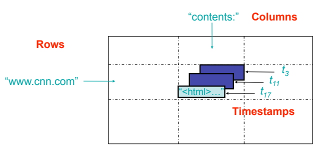
Also fun and challenging to build large-scale systems :)

Google

Basic Data Model

- Distributed multi-dimensional sparse map

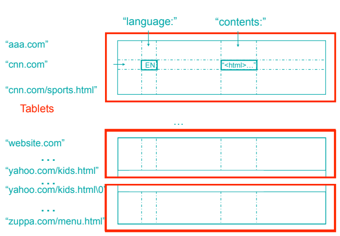
  (row, column, timestamp) → cell contents



- Rows are ordered lexicographically
- Good match for most of our applications

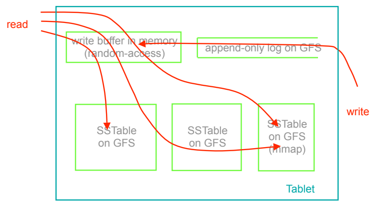
Google

Tablets & Splitting



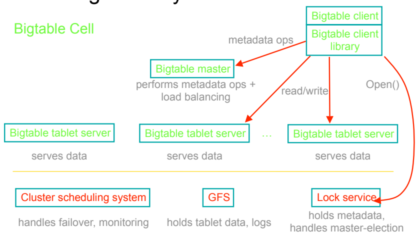
Google

Tablet Representation



SSTable: Immutable on-disk ordered map from string->string
string keys: *<row, column, timestamp>* triples

BigTable System Structure



BigTable Status

- Design/initial implementation started beginning of 2004
- Currently ~500 BigTable cells
- Production use or active development for ~70 projects:
  - Google Print
  - My Search History
  - Orkut
  - Crawling/indexing pipeline
  - Google Maps/Google Earth
  - Blogger
  - …
- Largest bigtable cell manages ~6000TB of data spread over several thousand machines (larger cells planned)

Future Infrastructure Directions

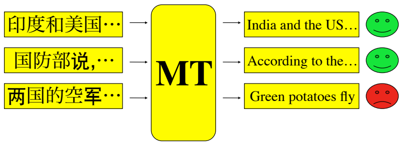
Existing systems mostly designed to work within cluster or datacenter

Current work: *Spanner*

- Next generation system that span all our datacenters
  - single global namespace
  - stronger consistency across datacenters
    - tricky in presence of partitions
  - allow higher-level constraints:
    - *"please keep this data on 2 disks in U.S., 2 in Europe and 1 in Asia"*
  - computational model to allow tying computation with underlying data
  - design goal: much more automated operation

Machine Translation

Goal: High quality translation of natural language text



Statistical Approach

Viewpoint of statistical Machine Translation (MT):

- Build probabilistic model of translation process
- Explore translation space to maximal prob. translated sentence, given source sentence

Main source of data for building statistical models:

- Parallel aligned corpora (text with sentence-by-sentence translations)
- Source and target language models (trained on huge amounts of text)
  - 5-gram target lang. model makes translations sound more natural

Try Chinese & Arabic translation systems at translate.google.com

Statistical Approach

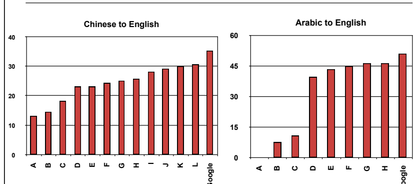
Language models trained on > 1000 billion words are **huge**

- 45.6 billion 5-grams
- 66.5% singletons: but, filtering rare events hurts Bleu score
- 1.5 terabyte of count data

Fun system design problems:

- Each sentence needs 100,000 to 1M language model lookups
- Language model doesn't fit on single machine: needs 100s of machines

NIST 2005 Results (%BLEU Score)



BLEU score: roughly fraction of multi-word phrase overlap with set of human translations

Official results at:
http://www.nist.gov/speech/tests/mt/mt05eval_official_results_release_20050801_v3.html

Example translation: Arabic - English
(non-Google translation service available on the web)

The Bradi : The inspectors need to “a few months” for end important their

Paris 13 - 1 (aa so in in) - the general manager for agency announced international for energy atomic Mohammed the Bradi today Monday that inspectors of international nze' the weapons need to “a few months” for end important their in Iraq.

Journalistic conference in end of meeting with French External Minister of Dominique de Villepin that the inspectors said during “a few their need to important months for end”.

...

The Bradi that Security Council confirmed “understands” that January 27 final term not.

...

Google

Example translation: Arabic - English
(Google System; 2005)

El Baradei : Inspectors Need "a Few Months" to Complete Their Mission

Paris 13 - 1 (AFP) - The Director General of the International Atomic Energy Agency Mohamed El Baradei announced today, Monday, that the international disarmament inspectors need "a few months" to complete their mission in Iraq.

He said during a press conference at the conclusion of a meeting with French Foreign Minister Dominique de Villepin that the inspectors "need a few months to complete their mission."

...

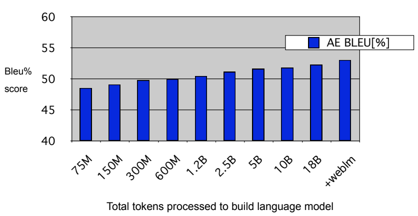
El Baradei stressed that the Security Council "understands" that the 27 January deadline is not final.

...

Google

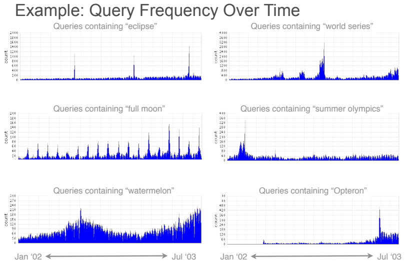
More Data is Better

Each doubling of LM training corpus size: ~0.5% higher BLEU score

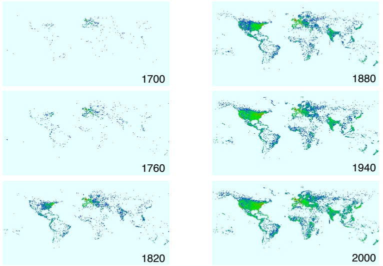


Google

Example: Query Frequency Over Time



Locations Mentioned in Books Over Time



Thanks to Matthew Gray

Google

Source Code Philosophy

- Google has one large shared source base
  - lots of lower-level libraries used by almost everything
  - higher-level app or domain-specific libraries
  - application specific code
- Many benefits:
  - improvements in core libraries benefit everyone
  - easy to reuse code that someone else has written in another context
- Drawbacks:
  - reuse sometimes leads to tangled dependencies
- Essential to be able to easily search whole source base
  - gsearch: internal tool for fast regexp searching of source code
  - huge productivity boost: easy to find uses, defs, examples, etc.
  - makes large-scale refactoring or renaming easier

Software Engineering Hygiene

- Code reviews
- Design reviews
- Lots of testing
  - unittests for individual modules
  - larger tests for whole systems
  - continuous testing system
- Most development done in C++, Java, & Python
  - C++: performance critical systems (e.g. everything for a web query)
  - Java: lower volume apps (advertising front end, parts of gmail, etc.)
  - Python: configuration tools, etc.

Multi-Site Software Engineering

- Google has moved from one to a handful to 20+ engineering sites around the world in last few years
- Motivation:
  - hire best canidates, regardless of their geographic location
- Issues:
  - more coordination needed
  - communication somewhat harder (no hallway conversations, time zone issues)
  - establishing trust between remote teams important
- Techniques:
  - online documentation, e-mail, video conferencing, careful choice of interfaces/project decomposition
  - example: BigTable project is split across three sites

Fun Environment for Software Engineering

- Very interesting problems
  - wide range of areas: low level hw/sw, dist. systems, storage systems, information retrieval, machine learning, user interfaces, auction theory, new product design, etc.
  - lots of interesting data and computational resources
- Service-based model for software development is very nice
  - very fluid, easy to make changes, easy to test, small teams can accomplish a lot
- Great colleagues/environment
  - expertise in wide range of areas, lots of interesting talks, etc.
- Work has a very large impact
  - hundreds of millions of users every month

Thanks! Questions...?

Further reading:

•Ghemawat, Gobioff, & Leung. *Google File System*, SOSP 2003.

•Barroso, Dean, & Hölzle . *Web Search for a Planet: The Google Cluster Architecture*, IEEE Micro, 2003.

•Dean & Ghemawat. *MapReduce: Simplified Data Processing on Large Clusters*, OSDI 2004.

•Chang, Dean, Ghemawat, Hsieh, Wallach, Burrows, Chandra, Fikes, & Gruber. *Bigtable: A Distributed Storage System for Structured Data*, OSDI 2006.

•Brants, Popat, Xu, Och, & Dean. *Large Language Models in Machine Translation*, EMNLP 2007.

These and many more available at: http://labs.google.com/papers.html

Google