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
- User interfaces
- Machine learning, Statistics, Information retrieval, AI
- Data structures, Algorithms
- Compilers, Programming languages
- Networking, Distributed systems, Fault tolerance
- Hardware, Mechanical engineering

- and much, much more!

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 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, heat machines in 15 mins, >10 days to recover)
- ~1 PDU failure (~500-1000 machines suddenly disappear, ~1 hour to recover)
- ~1 rack move (plenty of warning, ~500-1000 machines powered down, ~6 hours)
- ~1 rack failure (taking ~40% of machines down over 2 day span)
- ~20 lock failures (cold machines instantly disappear, 1/4 hours to get back)
- ~6 rack failures (50-400 machines over 6-7 days)
- ~4 network maintenance (it might cause ~30 minutes outages connectivity issues)
- ~20 network issues (takes an hour and everyone up for a couple minutes)
- ~5 router failures (have to immediately pull traffic for an hour)
- ~40-50 individual machine failures
- ~40-50 individual machine failures
- ~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 legal scales, super-thinly reliable hardware still fails, albeit less often
  - software still needs to be fault-tolerant
    - commodity machines without basic hardware fail often

Reliability has to come from the software

How can we make it easy to write distributed programs?
Overview of Infrastructure

- GFS, MapReduce, BigTable

- A peek at machine translation
- Some fun with interesting data
- General software engineering style/philosophy

Rest of Talk

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

GFS Design

- 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)

GFS Usage @ Google

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

Map-Reduce Programming Model
Code Example

```cpp
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

**Input**
- data

**Map**
- input to all overlapping latitude-longitude rectangles

**Shuffle**
- sort by key (key = Rect. Id)

**Reduce**
- render tile using data for all enclosed features

**Output**
- rendered tiles

Geographic feature list
- I-5
- Lake Washington
- WA-520
- I-90
- ... (N, I-5)
- ... (S, I-90)

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

Parallel MapReduce

**Input data**

**Master**

**Partitioned output**
Example Status

Reducing

Mapping / Shuffling

MapReduce Programs in Google’s Source Tree

New MapReduce Programs Per Month

Usage Statistics Over Time

<table>
<thead>
<tr>
<th></th>
<th>Aug. '04</th>
<th>Mar. '05</th>
<th>Mar. '06</th>
<th>Sep. '07</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of jobs</td>
<td>296</td>
<td>720</td>
<td>711</td>
<td>2,217</td>
</tr>
<tr>
<td>Average completion time</td>
<td>634</td>
<td>934</td>
<td>874</td>
<td>395</td>
</tr>
<tr>
<td>(secs)</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Machine years used</td>
<td>217</td>
<td>981</td>
<td>2,002</td>
<td>11,081</td>
</tr>
<tr>
<td>Input data read (TB)</td>
<td>3,288</td>
<td>12,571</td>
<td>52,254</td>
<td>403,152</td>
</tr>
<tr>
<td>Intermediate data (TB)</td>
<td>758</td>
<td>2,756</td>
<td>6,743</td>
<td>34,774</td>
</tr>
<tr>
<td>Output data written (TB)</td>
<td>193</td>
<td>941</td>
<td>2,970</td>
<td>14,018</td>
</tr>
<tr>
<td>Average worker machines</td>
<td>157</td>
<td>232</td>
<td>258</td>
<td>394</td>
</tr>
</tbody>
</table>
• Lots of (semi-)structured data at Google
  – URLs:
    • Contents, crawl metadata, links, anchors, pagerank, …
    • Per-user data:
      • User preference settings, recent query/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

BigTable: Motivation
• 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 :)

Why not just use commercial DB?
• Distributed multi-dimensional sparse map
  (row, column, timestamp) → cell contents
  • Rows are ordered lexicographically
  • Good match for most of our applications

Basic Data Model

Tablets & Splitting
Tablet Representation

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

BigTable System Structure

BigTable Cell

BigTable Client

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
- sticky 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
- 95.5% 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
El Baradei: Inspectors Need "a Few Months" to Complete Their Mission

Paris - (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 said during "a few more need to important months for end."

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

Example translation: Arabic - English

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

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

Mohammed the Bradi said as a result of meeting with French General 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 "understand" that January 27 final month.

Example translation: Arabic - English

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

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

(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.

Example: Query Frequency Over Time

Queries containing "world series"

Queries containing "eclipse"

Queries containing "summer olympics"

Queries containing "Opteron"

Queries containing "full moon"

Queries containing "watermelon"

Jan 02 ʻ

Jul ʻ03

Jan ʻ02

Jul ʻ03

...
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 regex searching of source code
  - huge productivity boost: easy to find uses, def's, 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 candidates, 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 interface/project decomposition
  - example: BigTable project in 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:
- Dean, Ghemawat, Wallach, Weikum, Chandra, Fikes, & Gruber. A Distributed Storage System for Structured Data, OSDI 2006.
- These and many more available at: http://labs.google.com/papers.html