

## Google: A Computer Scientist's Playground

**Jochen Hollmann**  
Google Zürich und Trondheim  
joho@google.com



## Outline

- Mission, data, and scaling
- Systems infrastructure
- Parallel programming model: MapReduce
- Googles work environment



## Google's Mission

To organize the world's information  
and make it universally  
accessible and useful



## A sense of scale

### Example: The Web

- 20+ billion web pages x 20KB = 400+ terabytes
- One computer can read 30-35 MB/sec from disk
  - ~four month to read the web
- ~1,000 hard drives just to store the web
- Even more to do something with the data



## Dealing with Scale

- Hardware, networking
  - Building a basic computing platform with low cost
- Distributed systems
  - Building reliable systems out of many individual computers
- Algorithms, data structures
  - Processing data efficiently, and in new and interesting ways
- Machine learning, information retrieval
  - Improving quality of search results by analyzing (lots of) data
- User interfaces
  - Designing effective interfaces for search and other products
- Many others...



## Why Use Commodity PCs?

- Single high-end 8-way Intel server:
  - IBM eserver xSeries 440
  - 8 2-GHz Xeon, 64 GB RAM, 8 TB of disk
  - \$758,000
- Commodity machines:
  - Rack of 88 machines
  - 176 2-GHz Xeons, 176 GB RAM, ~7 TB of disk
  - \$278,000
- 1/3X price, 22X CPU, 3X RAM, 1X disk

Sources: racksaver.com, TPC-C performance results, both from late 2002



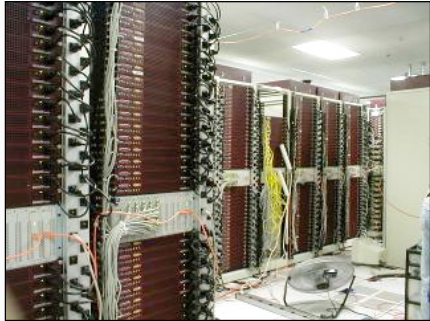
## google.stanford.edu (circa 1997)



## google.com (1999)



Google Data Center (circa 2000)



Google

google.com (new data center 2001)



Google

google.com (3 days later)



Google

### Implications of our Computing Environment

#### *Single-thread performance doesn't matter*

- We have large problems and total throughput/\$ more important than peak performance

#### *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/\$

**Fault-tolerant software makes cheap hardware practical**

Google

### An Example: The Index

- Similar to index in the back of a book (but big!)
  - Building takes several days on hundreds of machines
  - More than 4 billion web documents
  - Images: 880M images
  - File types: More than 35M non-HTML documents (PDF, Microsoft Word, etc.)
  - Usenet: 800M messages from >35K newsgroups

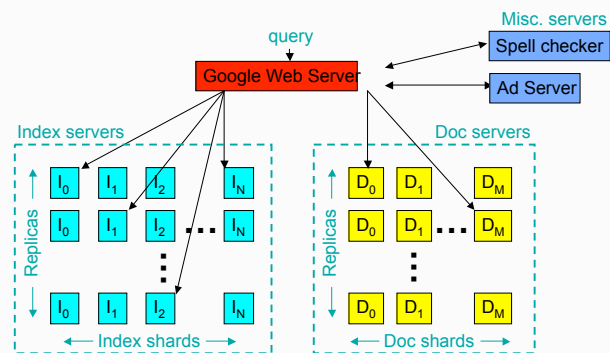


### Structuring the Index

- Too large for one machine, so...
  - Use PageRank as a total order
  - Split it into pieces, called **shards**, small enough to have several per machine
  - Replicate the shards, making more replicas of high PageRank shards
  - Do the same for the documents
  - Then: replicate this whole structure within and across data centers



### Google Query Serving Infrastructure



Elapsed time: 0.25s, machines involved: 1000+



### Reliable Building Blocks

- Need to store data reliably
- Need to run jobs on pools of machines
- Need to make it easy to apply lots of computational resources to problems

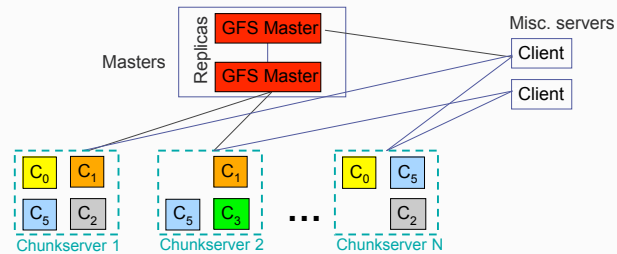
In-house solutions:

- Storage: Google File System (GFS)
- Job scheduling: Global Work Queue (GWQ)
- MapReduce: simplified large-scale data processing





## Google File System (GFS)



- Master manages metadata
- Data transfers happen directly between clients/chunkservers
- Files broken into chunks (typically 64 MB)
- Chunks triplicated across three machines for safety



## GFS Usage at Google

- 30+ Clusters
- Clusters as large as 2000+ chunkservers
- Petabyte-sized filesystems
- 2000+ MB/s sustained read/write load
- All in the presence of HW failures
- More information can be found in SOSP, '03



## MapReduce: Easy-to-use Cycles

- 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 optimization
  - 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



## More specifically...

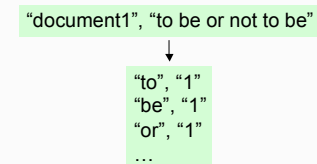
- Programmer specifies two primary methods:
  - `map(k, v) → <k', v'>*`
  - `reduce(k', <v'>*) → <k', v">*`
- All v' with same k' are reduced together, in order.
- Usually also specify:
  - `partition(k', total partitions) → partition for k'`
    - often a simple hash of the key
    - allows reduce operations for different k' to be parallelized



## Example: Word Frequencies in Web Pages

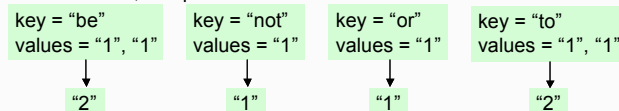
*A typical exercise for a new engineer in his or her first week*

- Input is files with one document per record
- Specify a `map` function that takes a key/value pair
  - key = document URL
  - value = document contents
- Output of map function is (potentially many) key/value pairs. In our case, output (word, "1") once per word in the document

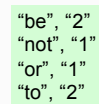


## Example continued: word frequencies in web pages

- MapReduce library gathers together all pairs with the same key (shuffle/sort)
- The `reduce` function combines the values for a key. In our case, compute the sum



- Output of reduce (usually 0 or 1 value) paired with key and saved



## Example: Pseudo-code

```

Map(String input_key, String input_value):
    // input_key: document name
    // input_value: document contents
    for each word w in input_values:
        EmitIntermediate(w, "1");

Reduce(String key, Iterator intermediate_values):
    // key: a word, same for input and output
    // intermediate_values: a list of counts
    int result = 0;
    for each v in intermediate_values:
        result += ParseInt(v);
    Emit(AsString(result));
  
```

Total 80 lines of C++ code including comments, main()



## Widely applicable at Google

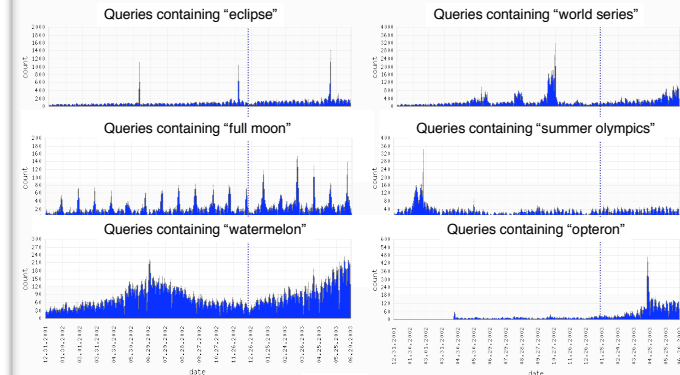
- Implemented as a C++ library linked to user programs
- Can read and write many different data types

### Example uses:

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



## Query Frequency Over Time



## Example: Generating Language Model Statistics

- Used in our statistical machine translation system
  - need to count # of times every 5-word sequence occurs in large corpus of documents (and keep all those where count  $\geq 4$ )
- Easy with MapReduce:
  - **map**: extract 5-word sequences => count from document
  - **reduce**: combine counts, and keep if count large enough



## Example: Joining with Other Data

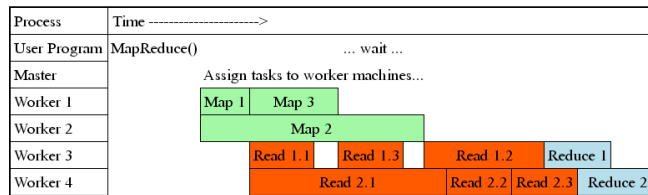
- Example: generate per-doc summary, but include per-host information (e.g. # of pages on host, important terms on host)
  - per-host information might be in per-process data structure, or might involve RPC to a set of machines containing data for all sites
- **map**: extract host name from URL, lookup per-host info, combine with per-doc data and emit
- **reduce**: identity function (just emit key/value directly)





## Task Granularity and Pipelining

- Fine granularity tasks: many more map tasks than machines
  - Minimizes time for fault recovery
  - Can pipeline shuffling with map execution
  - Better dynamic load balancing
- Often use 200,000 map/5000 reduce tasks w/ 2000 machines



### MapReduce status: MR\_Indexer-beta6-large-2003\_10\_28\_00\_03

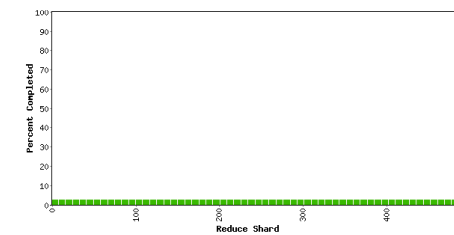
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 00 min 18 sec

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-	506631



### MapReduce status: MR\_Indexer-beta6-large-2003\_10\_28\_00\_03

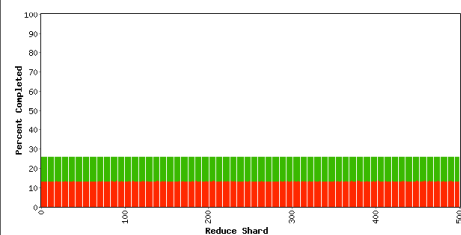
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 05 min 07 sec

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-	17290135



### MapReduce status: MR\_Indexer-beta6-large-2003\_10\_28\_00\_03

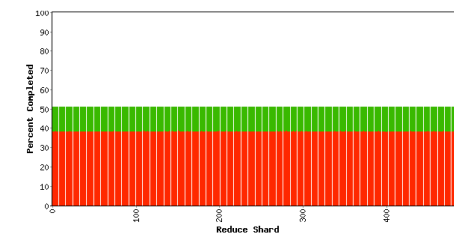
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 10 min 18 sec

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-	17300709

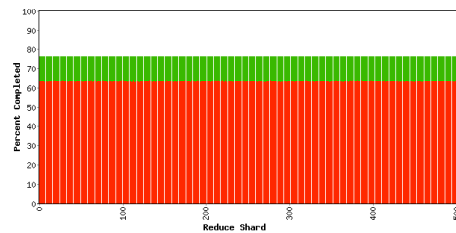


## MapReduce status: MR\_Indexer-beta6-large-2003\_10\_28\_00\_03

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

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

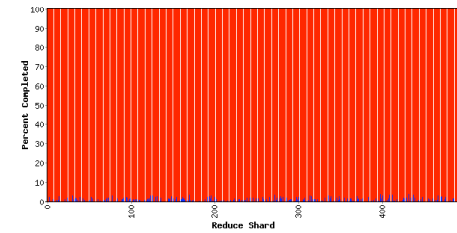


## MapReduce status: MR\_Indexer-beta6-large-2003\_10\_28\_00\_03

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

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

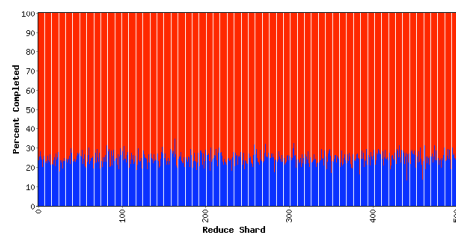


## MapReduce status: MR\_Indexer-beta6-large-2003\_10\_28\_00\_03

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

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

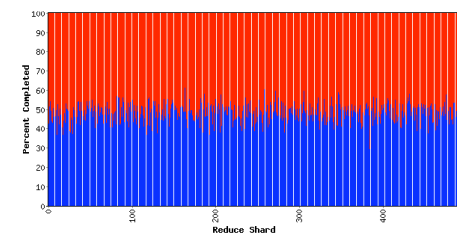


## MapReduce status: MR\_Indexer-beta6-large-2003\_10\_28\_00\_03

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

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



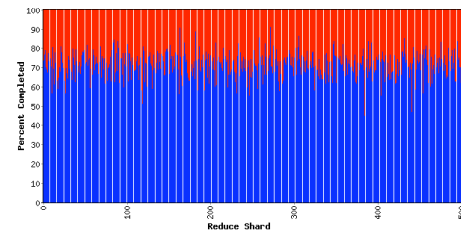


## MapReduce status: MR\_Indexer-beta6-large-2003\_10\_28\_00\_03

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

Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	1222.0
doc-index-hits	0 10
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	51640600
mr-merge-outputs	51640600

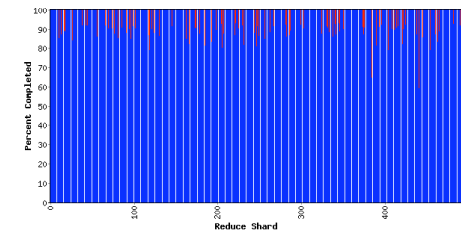


## MapReduce status: MR\_Indexer-beta6-large-2003\_10\_28\_00\_03

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

Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	849.5
doc-index-hits	0 10
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	35083350
mr-merge-outputs	35083350

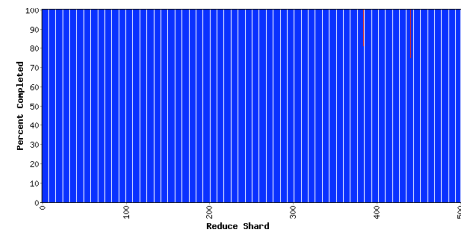


## MapReduce status: MR\_Indexer-beta6-large-2003\_10\_28\_00\_03

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

Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	9.4
doc-index-hits	0 10560
docs-indexed	0 36
dups-in-index-merge	0
mr-merge-calls	394792 36
mr-merge-outputs	394792 36

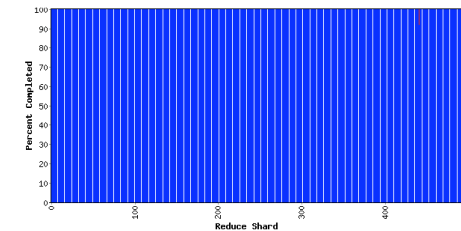


## MapReduce status: MR\_Indexer-beta6-large-2003\_10\_28\_00\_03

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

Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	1.9
doc-index-hits	0 10560
docs-indexed	0 36
dups-in-index-merge	0
mr-merge-calls	73442 36
mr-merge-outputs	73442 36



## Fault tolerance: Handled via re-execution

On worker failure:

- Detect failure via periodic heartbeats
- Re-execute completed and in-progress map tasks
- Re-execute in progress reduce tasks
- Task completion committed through master

On master failure:

- State is checkpointed to GFS: new master recovers & continues

Very Robust: lost 1600 of 1800 machines once, but finished fine



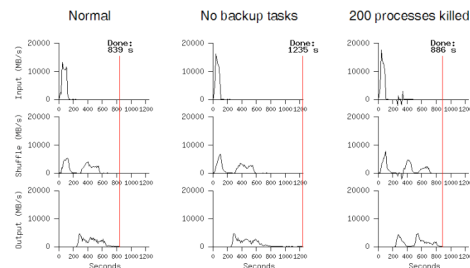
## Refinement: Backup Tasks

- Slow workers significantly lengthen completion time
  - Other jobs consuming resources on machine
  - Bad disks with soft errors transfer data very slowly
  - Weird things: processor caches disabled (!!)
- Solution: Near end of phase, spawn backup copies of tasks
  - Whichever one finishes first "wins"
- Effect: Dramatically shortens job completion time



## MR\_Sort

- Backup tasks reduce job completion time significantly
- System deals well with failures



## Refinement: Locality Optimization

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



## Refinement: Skipping Bad Records

Map/Reduce functions sometimes fail for particular inputs

- Best solution is to debug & fix, but not always possible

On seg fault:

- Send UDP packet to master from signal handler
- Include sequence number of record being processed

If master sees  $K$  failures for same record (typically  $K$  set to 2 or 3) :

- Next worker is told to skip the record

Effect: Can work around bugs in third-party libraries



## Other Refinements

- Optional secondary keys for ordering
- Compression of intermediate data
- Combiner: useful for saving network bandwidth
- Local execution for debugging/testing
- User-defined counters



## Performance Results & Experience

*Using 1,800 machines:*

- MR\_Grep scanned 1 terabyte in 100 seconds
- MR\_Sort sorted 1 terabyte of 100 byte records in 14 minutes

*Rewrote Google's production indexing system*

- a sequence of 7, 10, 14, 17, 21, 24 MapReductions
- simpler
- more robust
- faster
- more scalable



## Implications for Multi-core Processors

- Multi-core processors require parallelism, but many programmers are uncomfortable writing parallel programs
- MapReduce provides an easy-to-understand programming model for a very diverse set of computing problems
  - users don't need to be parallel programming experts
  - system automatically adapts to number of cores & machines available
- Optimizations useful even in single machine, multi-core environment
  - locality, load balancing, status monitoring, robustness, ...



## Data + CPUs = Playground

- Substantial fraction of internet available for processing
- Easy-to-use teraflops/petabytes
- Cool problems, great colleagues



## Google Product Suite

### Google Web Search

World's most used web search



### AdSense for Search

Monetizing search results pages



### AdSense for Content

Monetize content pages



### Blogger

Post your thoughts to the web



### Froogle: Product Search

20 M+ products from 150k+ sites



### Gmail

Web-based email with 1GB storage



### Google Deskbar

Search from the desktop at any time



### Google Search Appliance

Corporate search solution



### Google Toolbar

Search from browser, block popups, etc



## Who Does All This?

- Talented, motivated people
  - ... working in small teams (3-5 people)
  - ... on problems that matter
  - ... with freedom to explore their ideas
  - "20% rule", access to computational resources
- It's not just search: Google has experts in...
  - Hardware, networking, distributed systems, fault tolerance, data structures, algorithms, machine learning, information retrieval, AI, user interfaces, compilers, programming languages, statistics, product design, mechanical eng., ...



## Work philosophy: Let people work on exciting problems

- Work on things that **matter**
- Affect **everyone** in the world
- Solve problems with **algorithms** if at all possible
- Hire **bright** people and give them lots of **freedom**
- Don't be afraid to **try** new things
- Number 1 work place in the US and TOP 10 in Ireland

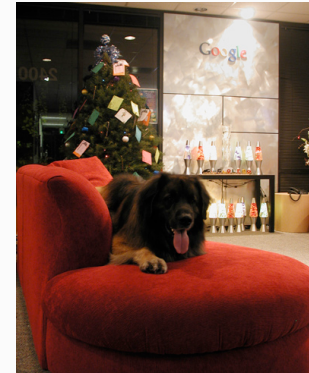


### Work Environment



Google

### Work Environment



Google

### Work Environment



Google

### Work Environment



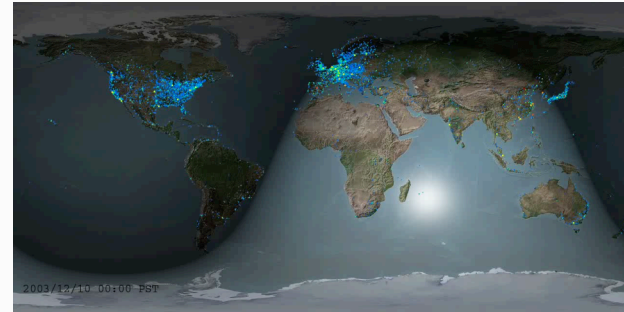
Google

## We're Hiring!

- If this sounds like fun, we have engineering positions in:  
Trondheim, Zürich, Munich, London, Dublin  
And around the globe!

<http://labs.google.com/why-google.html>

<http://google.com/jobs>



<http://labs.google.com/why-google.html>

<http://google.com/jobs>



## The End

- Thanks to Jeff Dean for compiling most of the slides

