Experiences with MapReduce, an Abstraction for Large-Scale Computation

Jeff Dean
Google, Inc.

Outline

- Overview of our computing environment
- MapReduce
  - overview, examples
  - implementation details
  - usage stats
- Implications for parallel program development

Google

Problem: lots of data

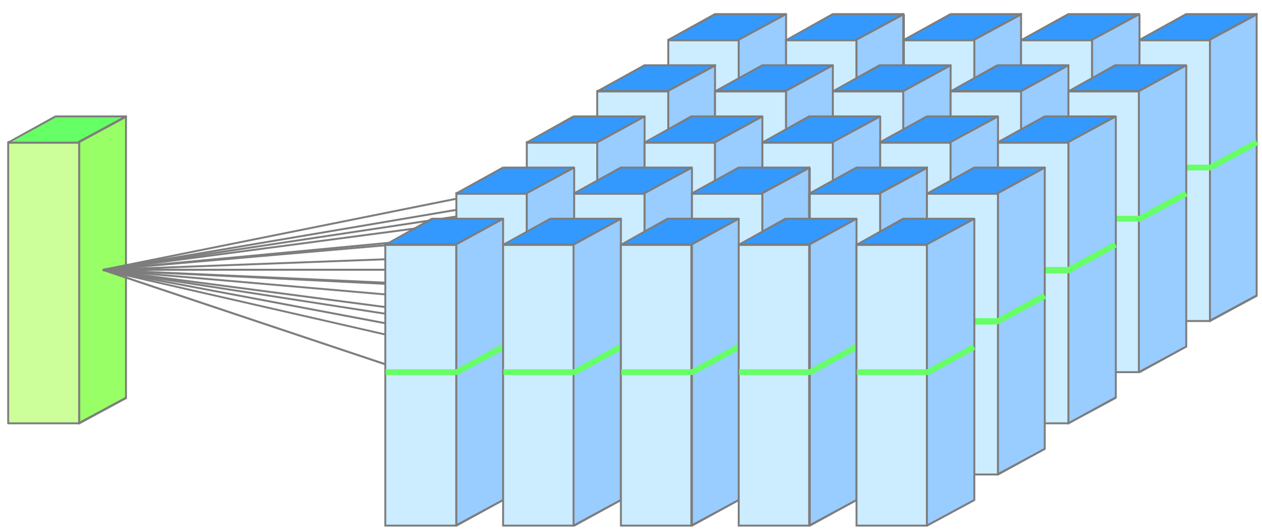
- Example: 20+ billion web pages x 20KB = 400+ terabytes
- One computer can read 30-35 MB/sec from disk
  - ~four months to read the web
- ~1,000 hard drives just to store the web
- Even more to *do* something with the data

Solution: spread the work over many machines

- Good news: same problem with 1000 machines, < 3 hours
- Bad news: programming work
  - communication and coordination
  - recovering from machine failure
  - status reporting
  - debugging
  - optimization
  - locality
- Bad news II: repeat for every problem you want to solve

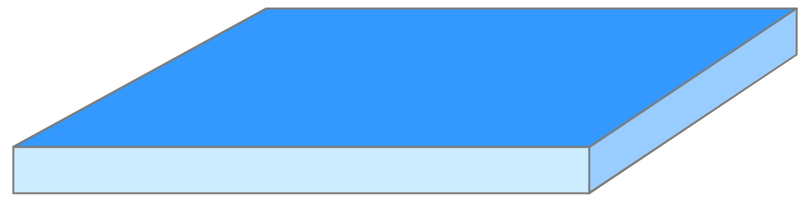
Computing Clusters

- Many racks of computers, thousands of machines per cluster
- Limited bisection bandwidth between racks



Machines

- 2 CPUs
  - Typically hyperthreaded or dual-core
  - Future machines will have more cores
- 1-6 locally-attached disks
  - 200GB to ~2 TB of disk
- 4GB-16GB of RAM
- Typical machine runs:
  - Google File System (GFS) chunkserver
  - Scheduler daemon for starting user tasks
  - One or many user tasks



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/$

How can we make it easy to write distributed programs?

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 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

“document1”, “to be or not to be”

↓

“to”, “1”
“be”, “1”
“or”, “1”
…

Google

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

key = “be” values = “1”, “1”	key = “not” values = “1”	key = “or” values = “1”	key = “to” values = “1”, “1”
↓	↓	↓	↓
“2”	“1”	“1”	“2”

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

“be”, “2”
“not”, “1”
“or”, “1”
“to”, “2”

Google

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()

Google

Widely applicable at Google

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

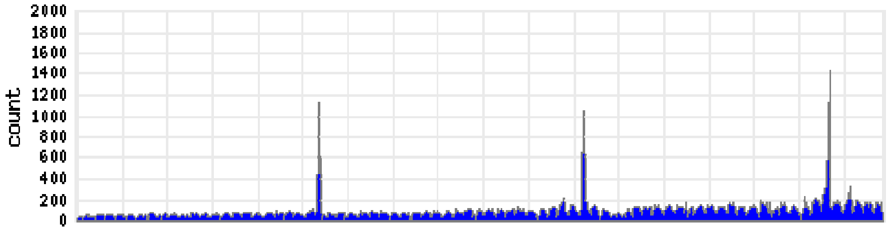
Example uses:

distributed grep
distributed sort
term-vector per host
document clustering
machine learning
...

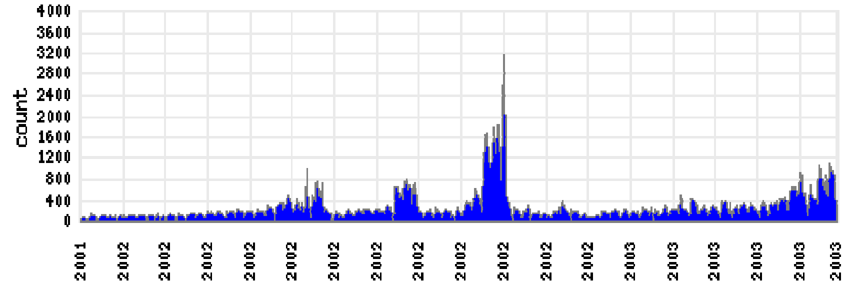
web access log stats
web link-graph reversal
inverted index construction
statistical machine translation
…

Example: Query Frequency Over Time

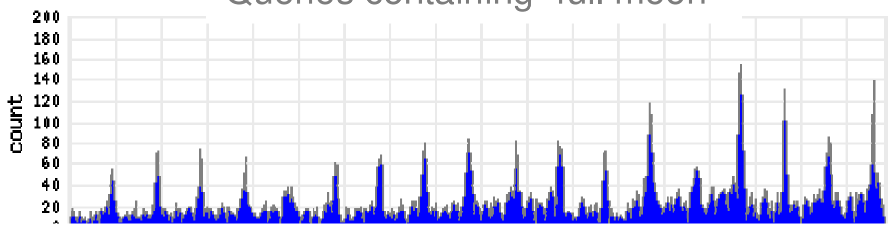
Queries containing “eclipse”



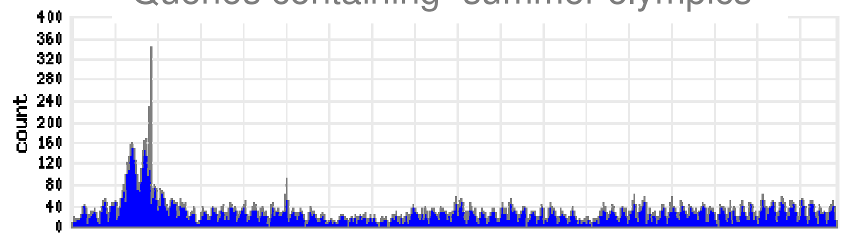
Queries containing “world series”



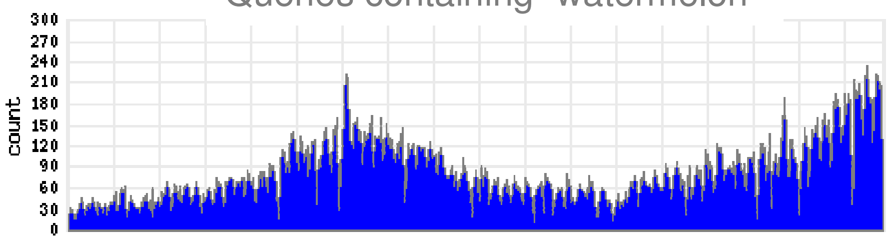
Queries containing “full moon”



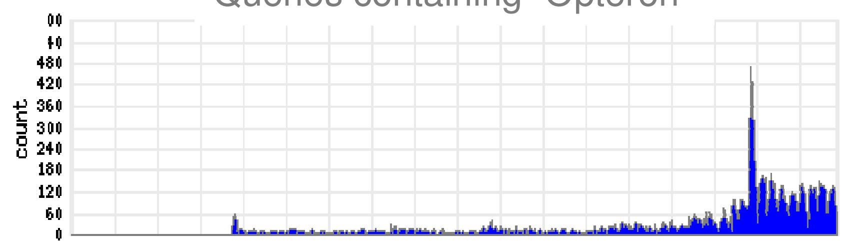
Queries containing “summer olympics”



Queries containing “watermelon”



Queries containing “Opteron”



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 >= 4)

- Easy with MapReduce:
  - map: extract 5-word sequences => count from document
  - reduce: combine counts, and keep if count large enough

Google

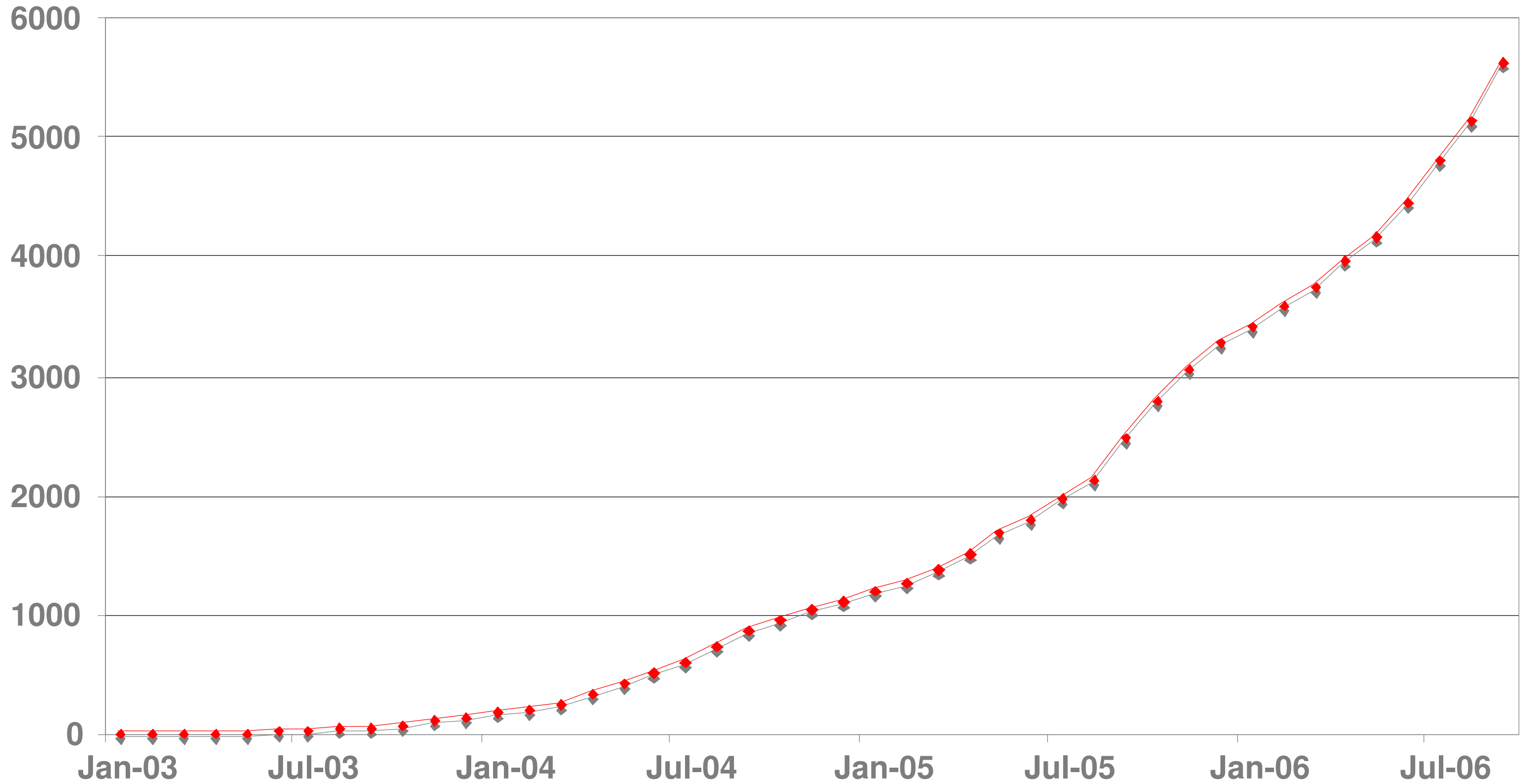
Example: Joining with Other Data

- 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)

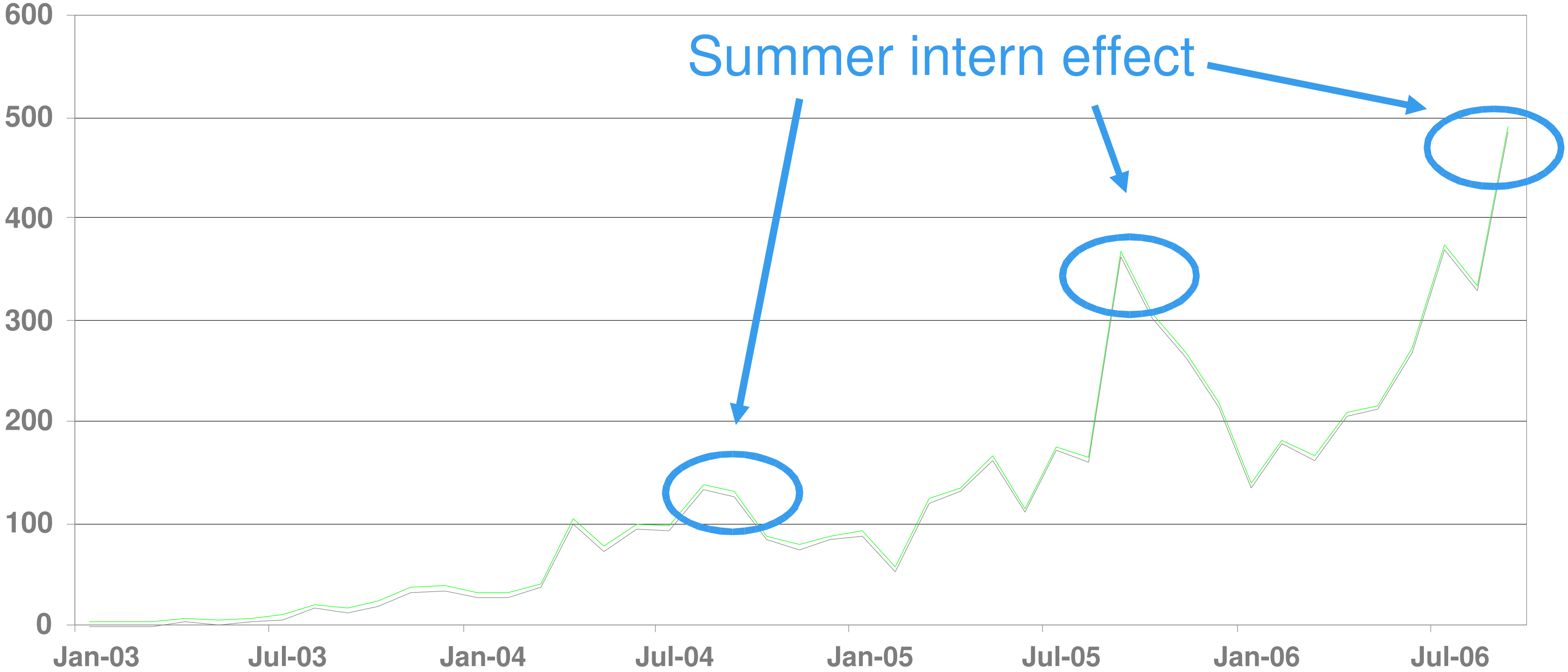
Google

MapReduce Programs in Google's Source Tree



Google

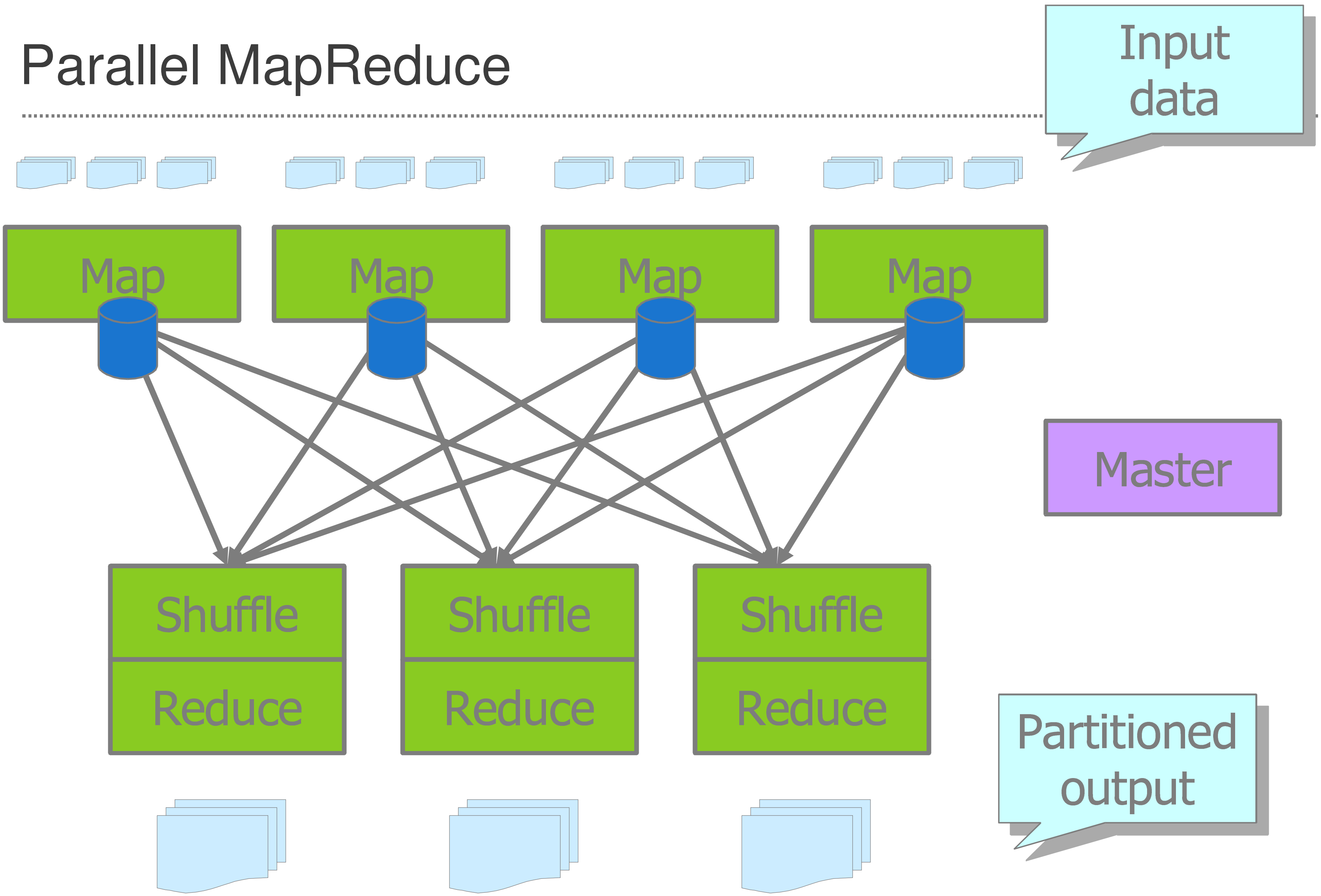
New MapReduce Programs Per Month



MapReduce: Scheduling

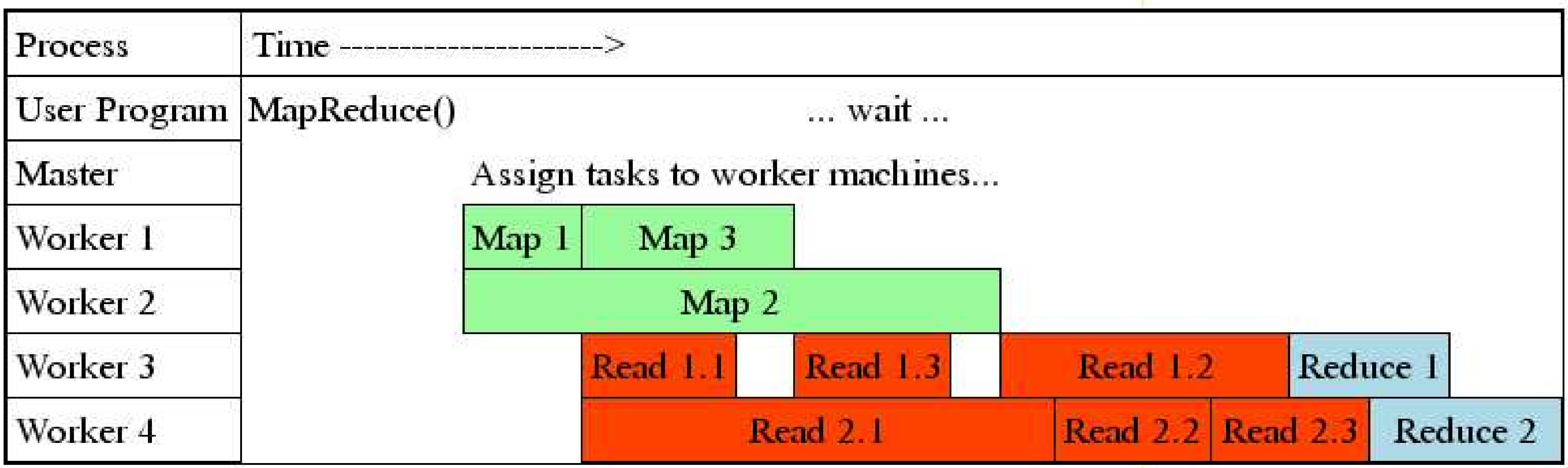
- **One master, many workers**
  - Input data split into *M* map tasks (typically 64 MB in size)
  - Reduce phase partitioned into *R* reduce tasks
  - Tasks are assigned to workers dynamically
  - Often: *M*=200,000; *R*=4,000; workers=2,000
- **Master assigns each map task to a free worker**
  - Considers locality of data to worker when assigning task
  - Worker reads task input (often from local disk!)
  - Worker produces R **local files** containing intermediate k/v pairs
- **Master assigns each reduce task to a free worker**
  - Worker reads intermediate k/v pairs from map workers
  - Worker sorts & applies user's *Reduce* op to produce the output

Parallel MapReduce



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

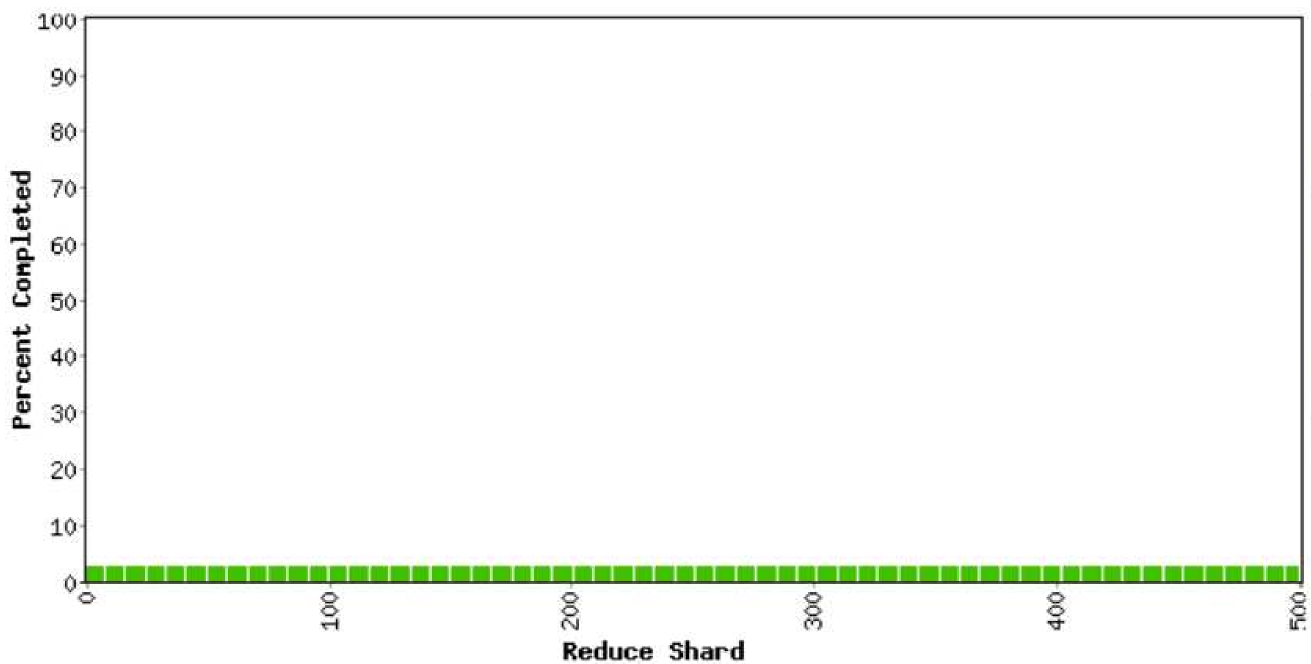


MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

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

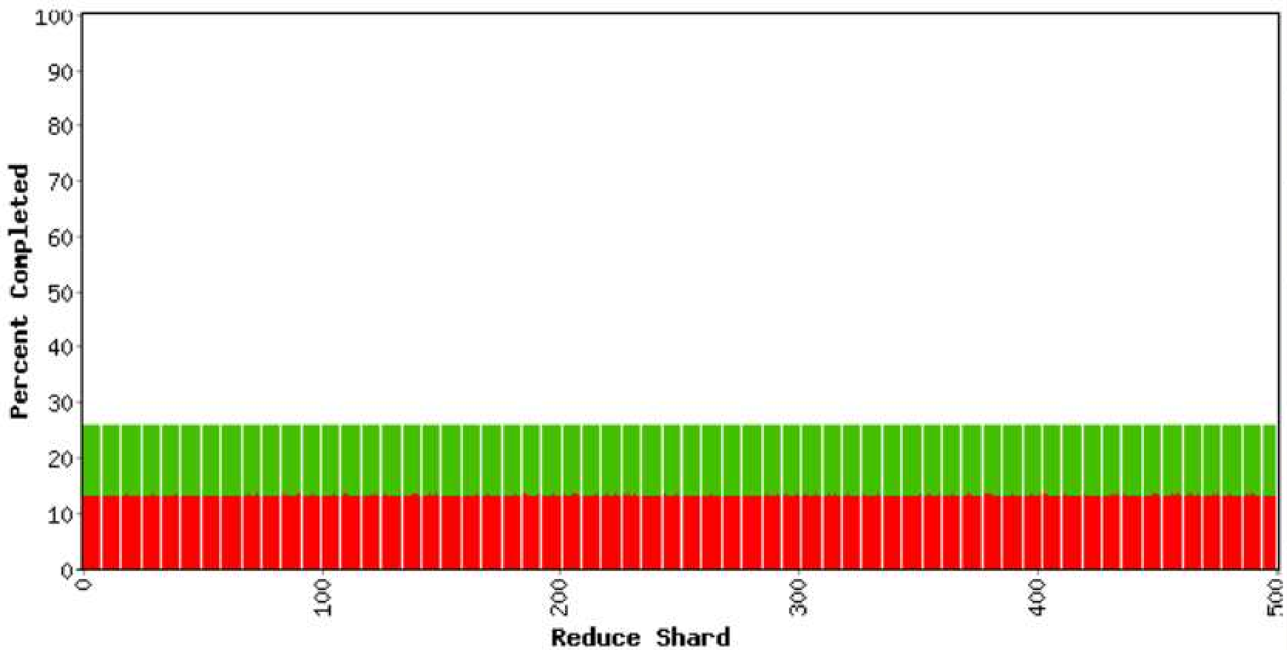
Variable	Minute
Mapped (MB/s)	72.5
Shuffle (MB/s)	0.0
Output (MB/s)	0.0
doc-index-hits	145825686
docs-indexed	506631
dups-in-index-merge	0
mr-operator-calls	508192
mr-operator-	506631

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

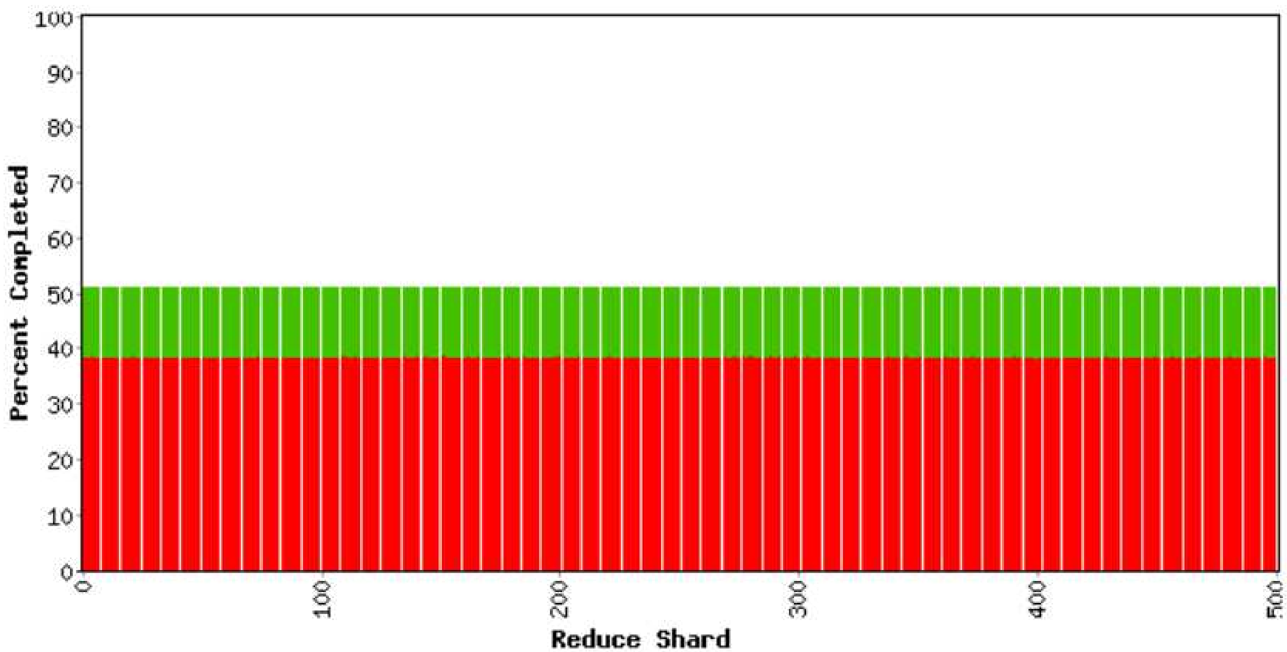
Variable	Minute
Mapped (MB/s)	699.1
Shuffle (MB/s)	349.5
Output (MB/s)	0.0
doc-index-hits	5004411944
docs-indexed	17290135
dups-in-index-merge	0
mr-operator-calls	17331371
mr-operator-outputs	17290135

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-outputs	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

Counters

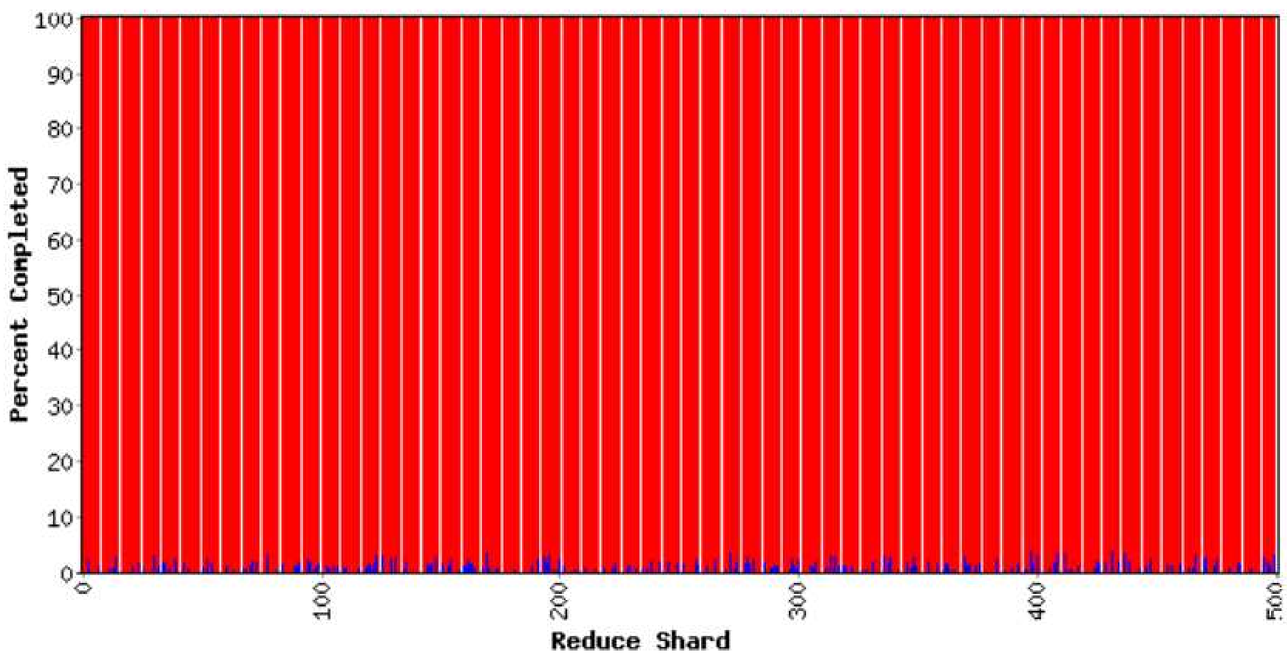
Variable	Minute
Mapped (MB/s)	706.5
Shuffle (MB/s)	419.2
Output (MB/s)	0.0
doc-index-hits	4982870667
docs-indexed	17229926
dups-in-index-merge	0
mr-operator-calls	17272056
mr-operator-outputs	17229926

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



Counters

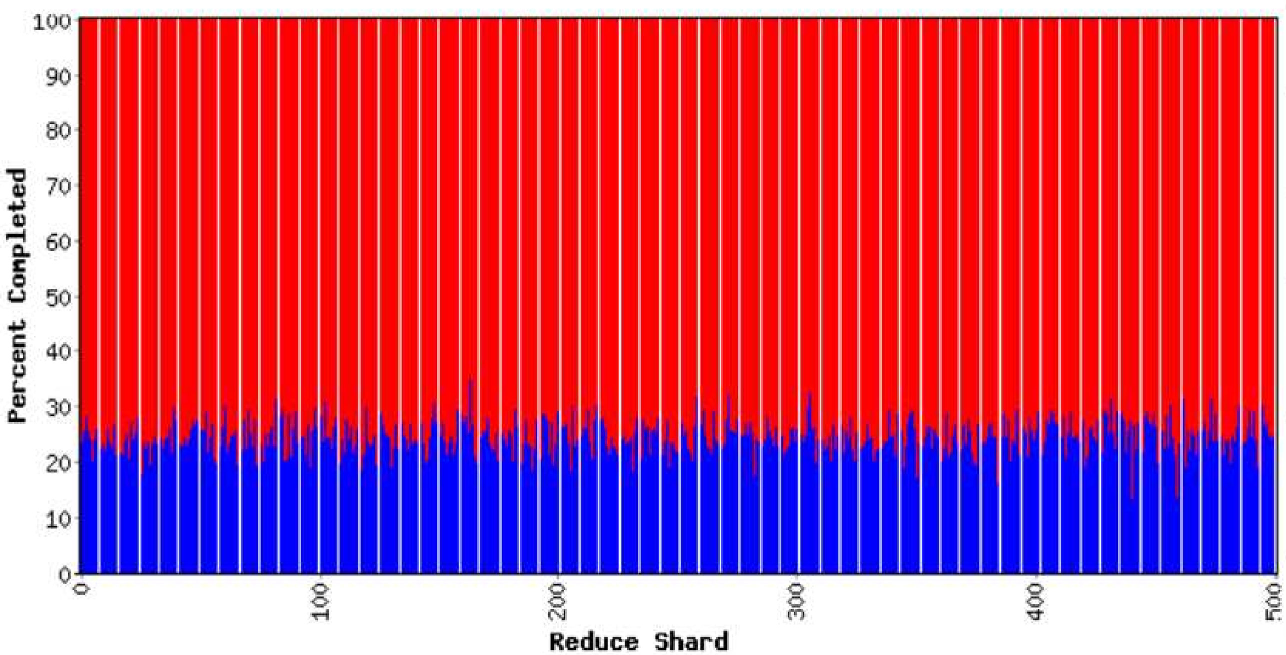
Variable	Minute	
Mapped (MB/s)	0.3	
Shuffle (MB/s)	0.5	
Output (MB/s)	45.7	
doc-index-hits	2313178	10
docs-indexed	7936	
dups-in-index-merge	0	
mr-merge-calls	1954105	
mr-merge-outputs	1954105	

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



Counters

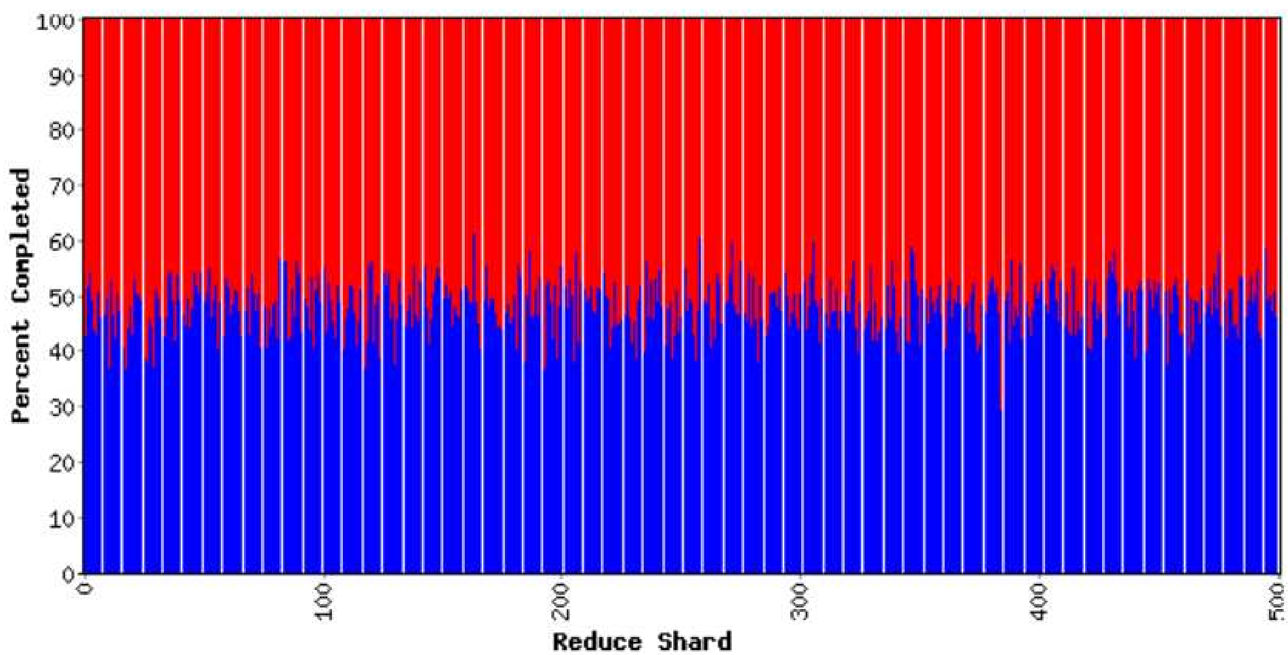
Variable	Minute
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.1
Output (MB/s)	1238.8
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	51738599
mr-merge-outputs	51738599

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



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1225.1	
doc-index-hits	0	1(
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	51842100	
mr-merge-outputs	51842100	

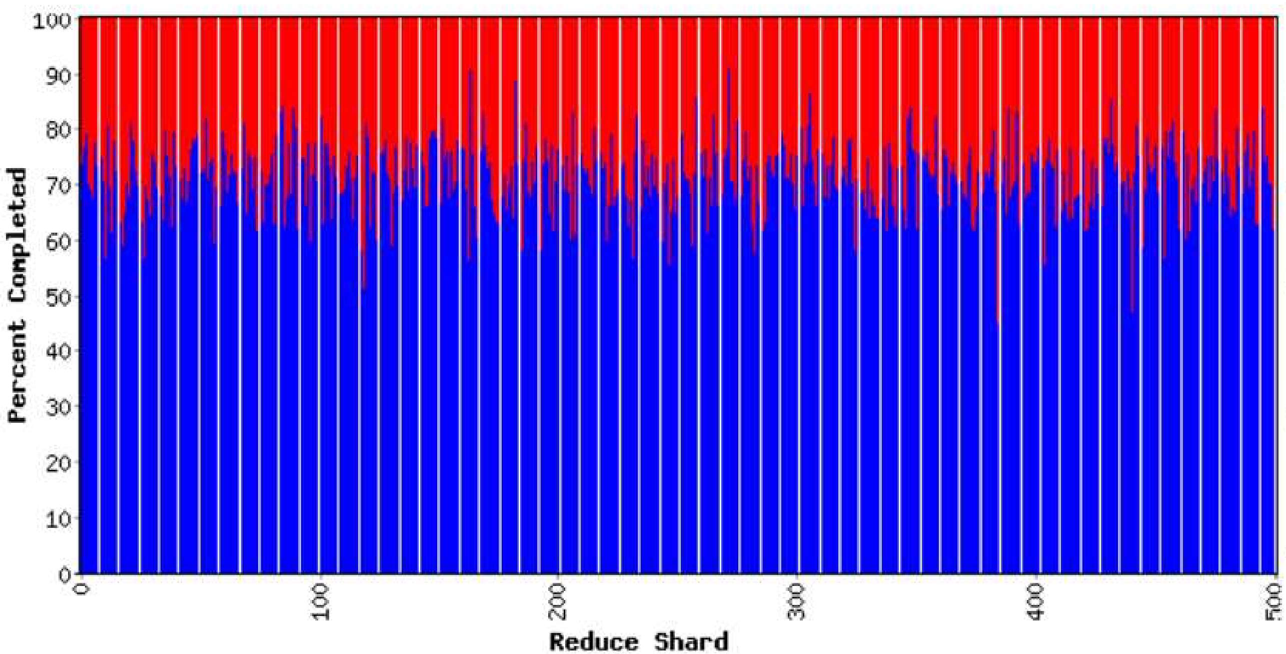
Google

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



Counters

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

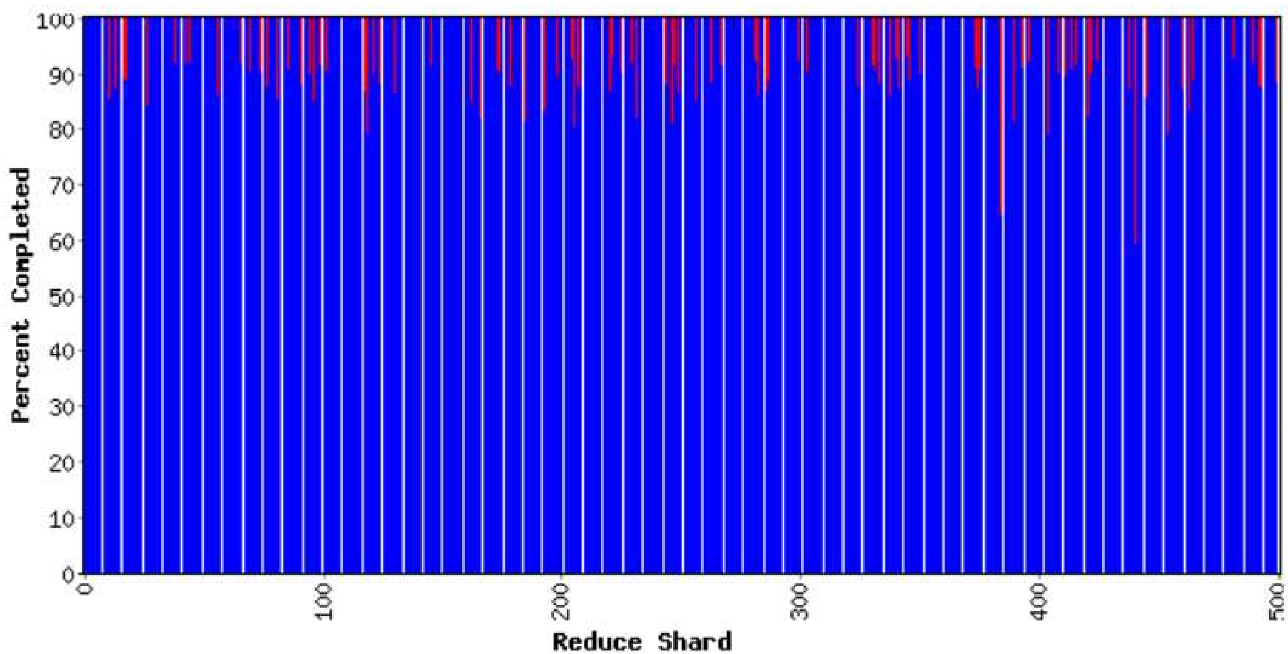
Google

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



Counters

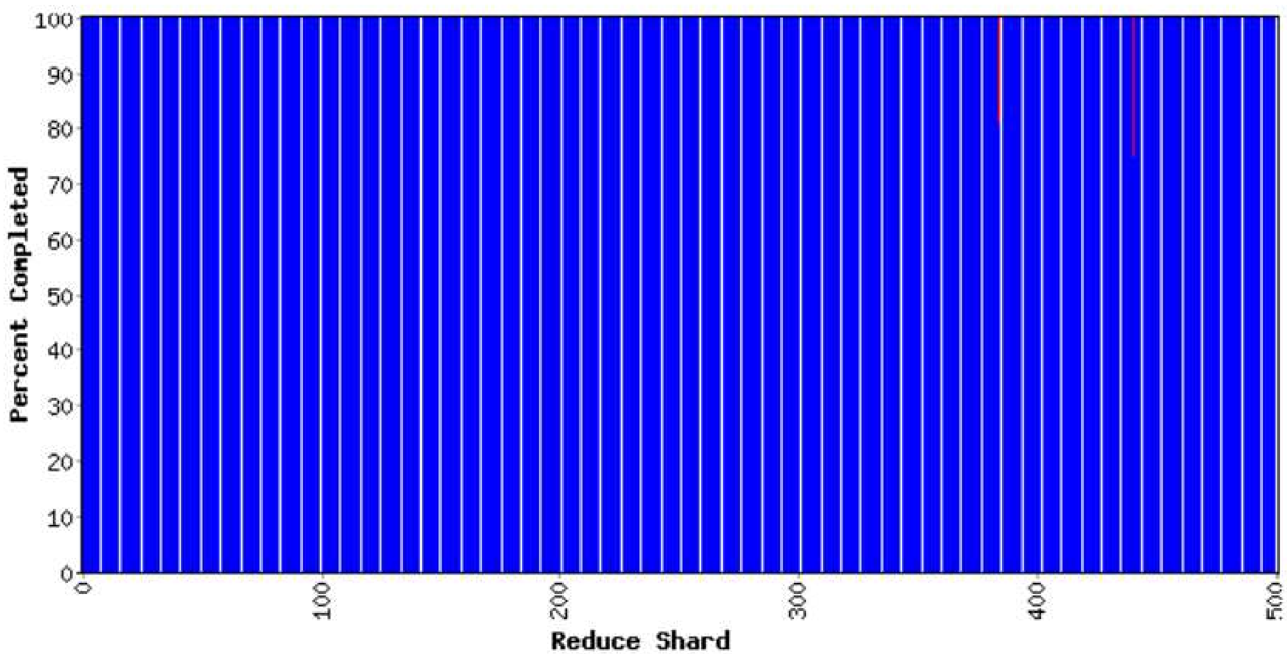
Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	849.5	
doc-index-hits	0	1
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	35083350	
mr-merge-outputs	35083350	

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



Counters

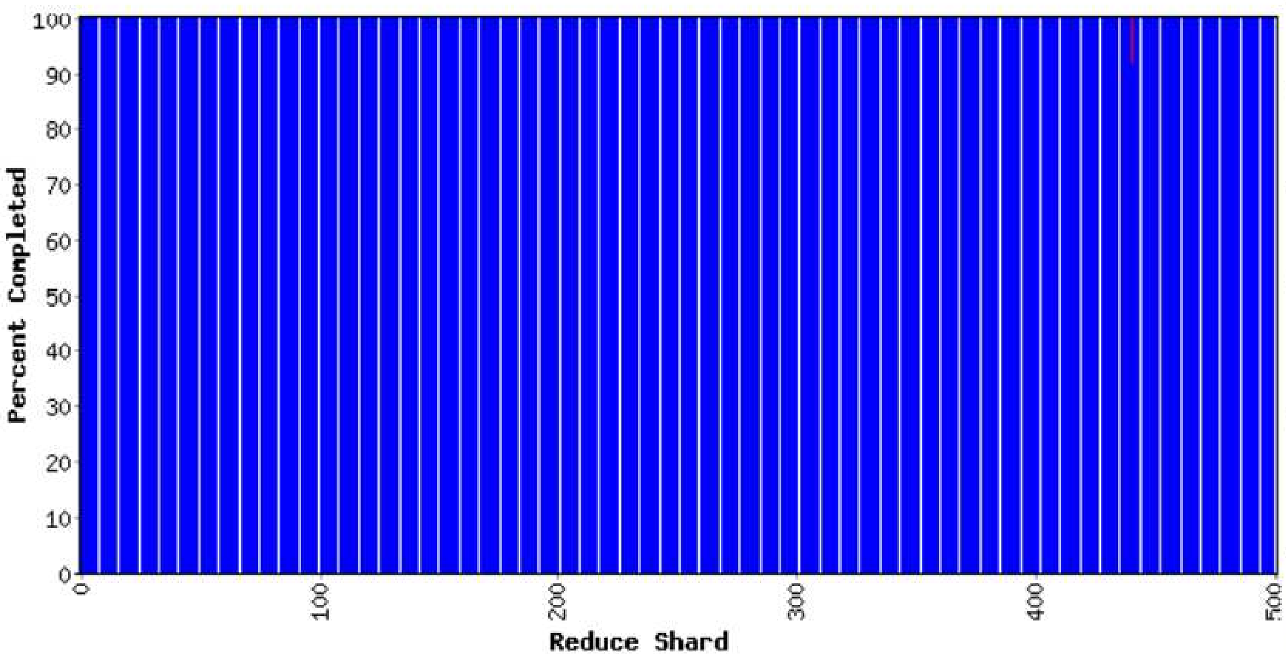
Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	9.4	
doc-index-hits	0	105
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	394792	
mr-merge-outputs	394792	

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



Counters

Variable	Minute	
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1.9	
doc-index-hits	0	105
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	73442	
mr-merge-outputs	73442	

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

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:

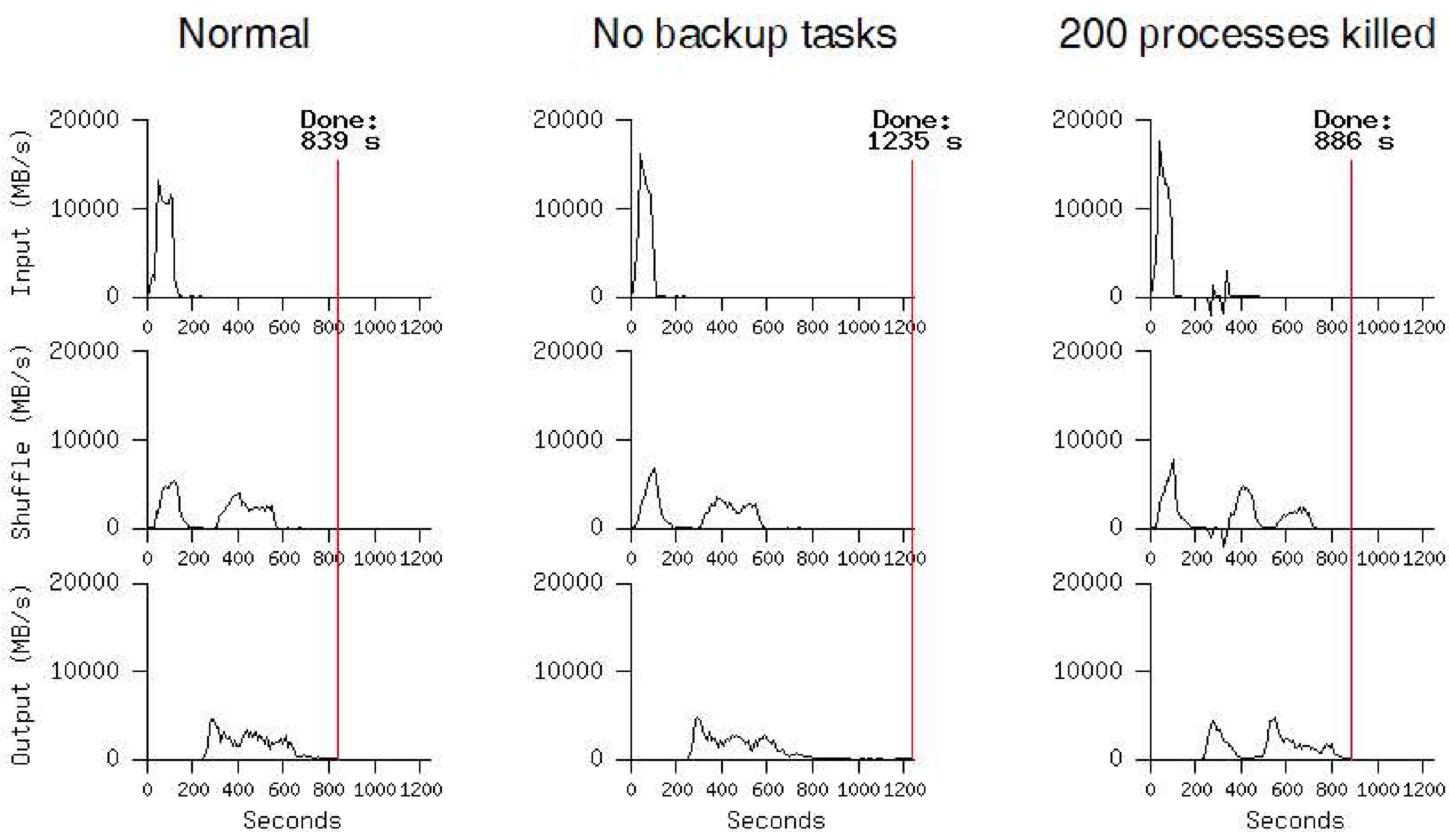
- MR_Grep scanned 1 terabyte in 100 seconds
- MR_Sort sorted 1 terabyte of 100 byte records in 14 minutes

Rewrote Google's production indexing system

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

MR_Sort

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



Usage Statistics Over Time

	Aug, '04	Mar, '05	Mar, '06
Number of jobs	29,423	72,229	171,834
Average completion time (secs)	634	934	874
Machine years used	217	981	2,002
Input data read (TB)	3,288	12,571	52,254
Intermediate data (TB)	758	2,756	6,743
Output data written (TB)	193	941	2,970
Average worker machines	157	232	268
Average worker deaths per job	1.2	1.9	5.0
Average map tasks per job	3,351	3,097	3,836
Average reduce tasks per job	55	144	147
Unique map/reduce combinations	426	411	2345

Google

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, …

Conclusion

- MapReduce has proven to be a remarkably-useful abstraction
- Greatly simplifies large-scale computations at Google
- Fun to use: focus on problem, let library deal with messy details
  - Many thousands of parallel programs written by hundreds of different programmers in last few years
  - Many had no prior parallel or distributed programming experience

Further info:

MapReduce: Simplified Data Processing on Large Clusters, Jeffrey Dean and Sanjay Ghemawat, OSDI'04

http://labs.google.com/papers/mapreduce.html

(or search Google for [MapReduce])