MapReduce: Simplified Data Processing on Large Clusters

PA154 Jazykové modelování (9.2)

Pavel Rychlý

pary@fi.muni.cz

April 27, 2021

Source: Jeff Dean, Sanjay Ghemawat
Google, Inc.
December, 2004
https://research.google/pubs/pub62/

Motivation: Large Scale Data Processing

Many tasks: Process lots of data to produce other data
Want to use hundreds or thousands of CPUs

- ... but this needs to be easy

MapReduce provides:

- Automatic parallelization and distribution
- Fault-tolerance
- I/O scheduling
- Status and monitoring

Programming model

Input & Output: each a set of key/value pairs Programmer specifies two functions:

```
map (in_key, in_value) -> list(out_key, intermediate_value)
```

- Processes input key/value pair
- Produces set of intermediate pairs

```
reduce (out_key, list(intermediate_value)) -> list(out_value)
```

- Combines all intermediate values for a particular key
- Produces a set of merged output values (usually just one)

Inspired by similar primitives in LISP and other languages

Example: Count word occurrences

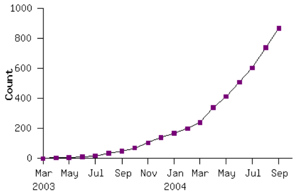
```
map(String input_key, String input_value):
  // input_key: document name
  // input_value: document contents
  for each word w in input_value:
    EmitIntermediate(w, "1");

reduce(String output_key, Iterator intermediate_values):
  // output_key: a word
  // output_values: a list of counts
  int result = 0;
  for each v in intermediate_values:
    result += ParseInt(v);
  Emit(AsString(result));
```

Pseudocode: See appendix in paper for real code

Model is Widely Applicable

MapReduce Programs In Google Source Tree



Example uses:

distributed grep
term-vector per host
document clustering
...

distributed sort
web access log stats
machine learning
...

web link-graph reversal
inverted index construction
statistical machine translation
...

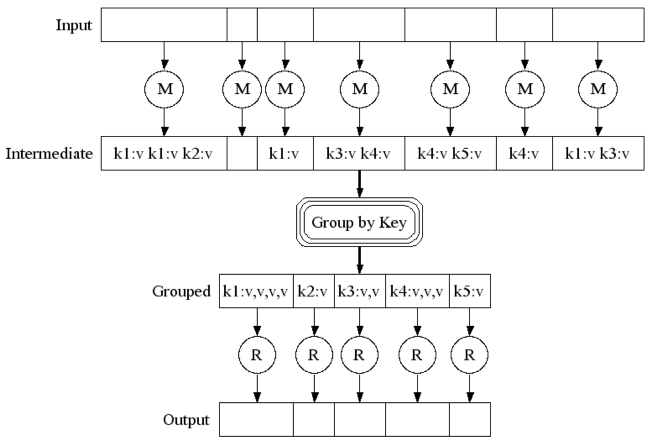
Implementation Overview

Typical cluster:

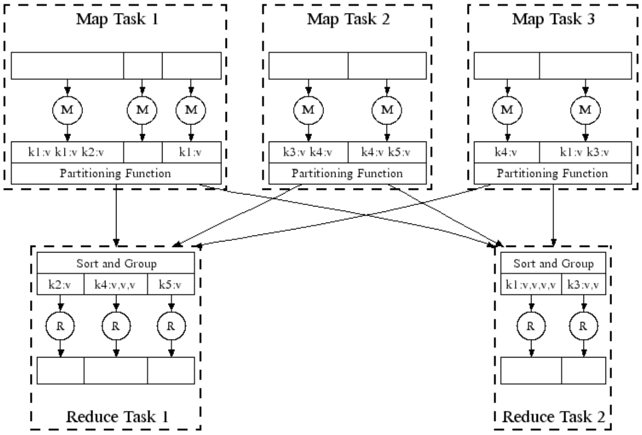
- 100s/1000s of 2-CPU x86 machines, 2-4 GB of memory
- Limited bisection bandwidth
- Storage is on local IDE disks
- GFS: distributed file system manages data (SOSP'03)
- Job scheduling system: jobs made up of tasks, scheduler assigns tasks to machines

Implementation is a C++ library linked into user programs

Execution



Parallel Execution



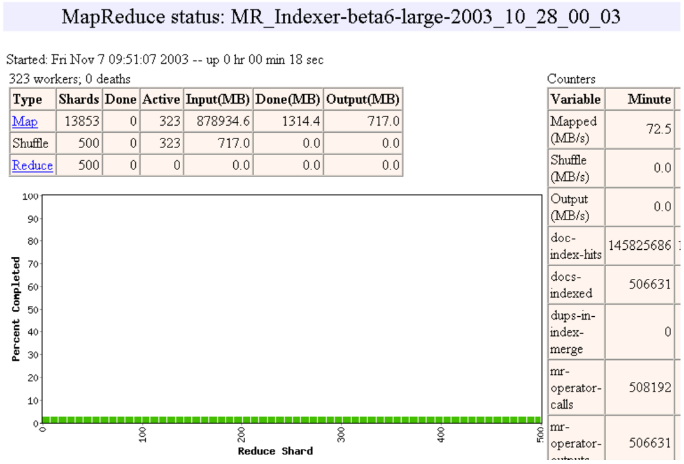
Task Granularity And Pipelining

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/ 2000 machines

Process	Time --------------------->
User Program	MapReduce() ... wait ...
Master	Assign tasks to worker machines...
Worker 1	Map 1 Map 3
Worker 2	Map 2
Worker 3	Read 1.1 Read 1.3 Read 1.2 Reduce 1
Worker 4	Read 2.1 Read 2.2 Read 2.3 Reduce 2



MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

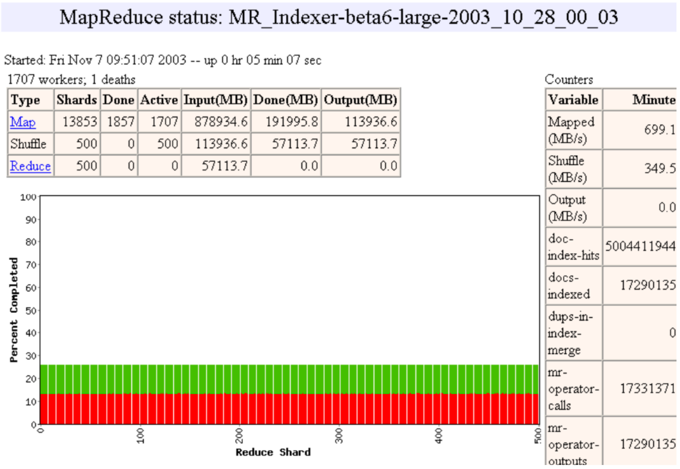
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 00 min 18 sec

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-outputs	506631



MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

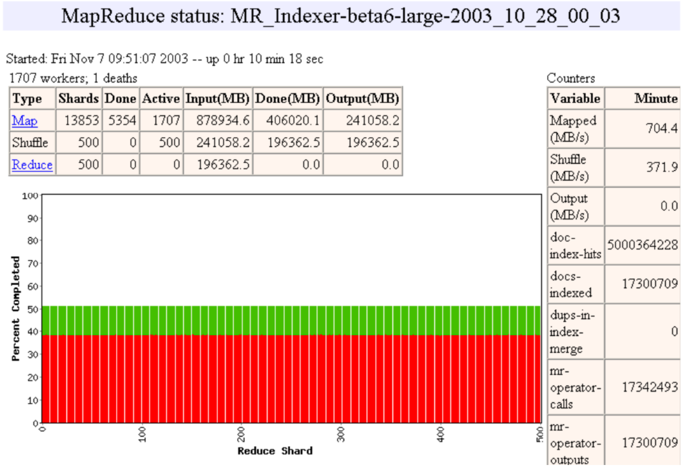
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 05 min 07 sec

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

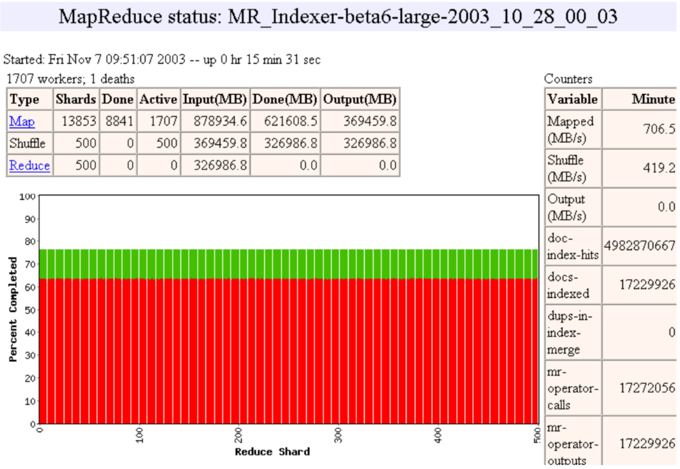
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 10 min 18 sec

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

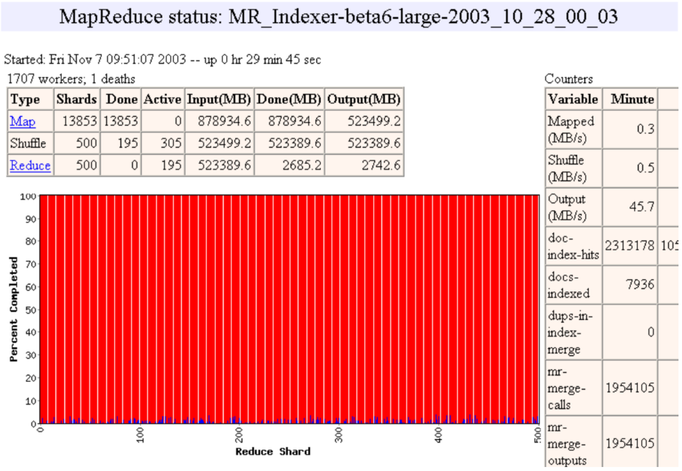
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 15 min 31 sec

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

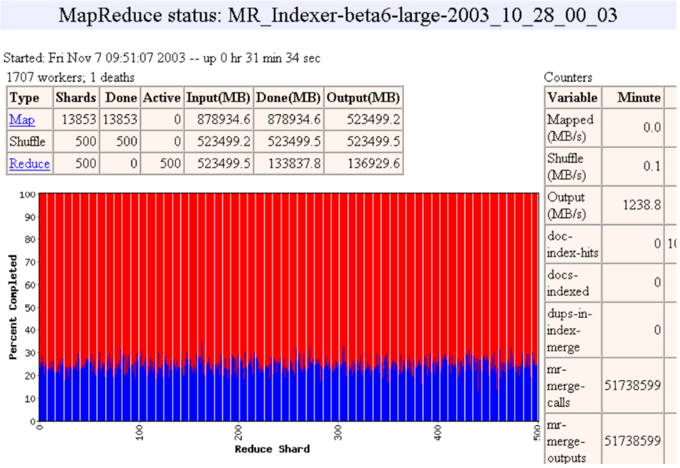
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 29 min 45 sec

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	105
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

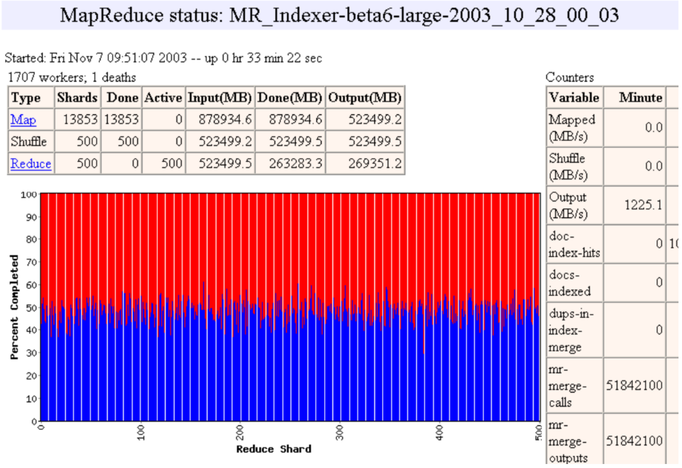
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 31 min 34 sec

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	10
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

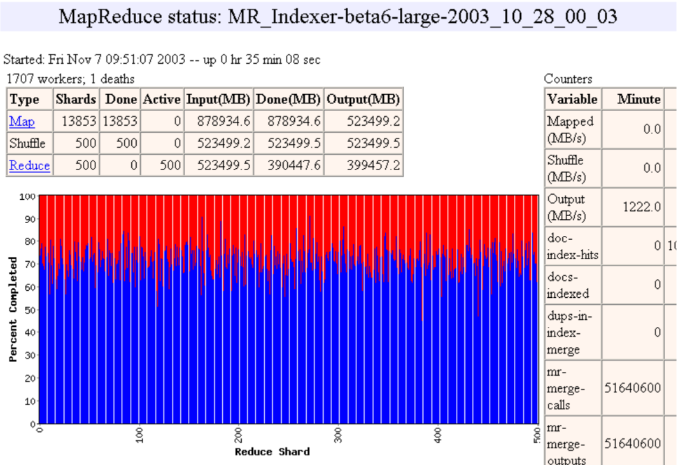
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 33 min 22 sec

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	10
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

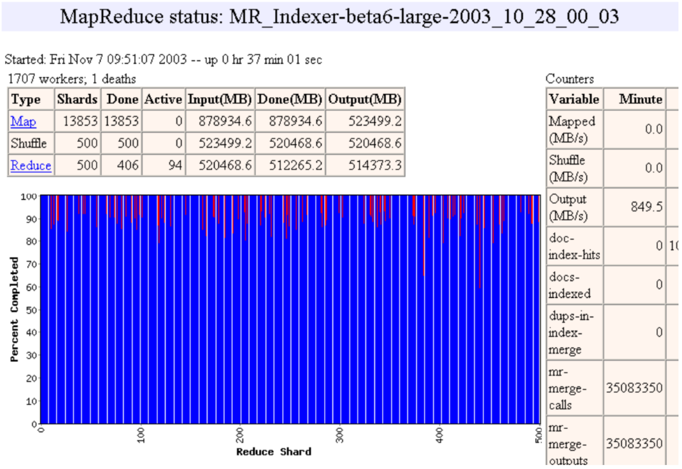
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 35 min 08 sec

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

Counters

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

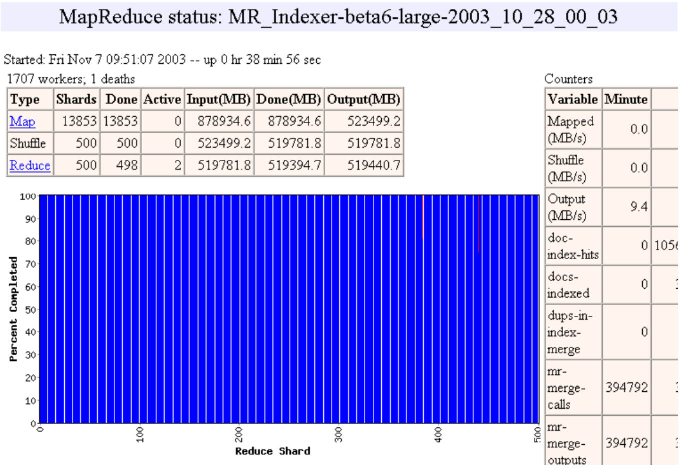
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 37 min 01 sec

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

Counters

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

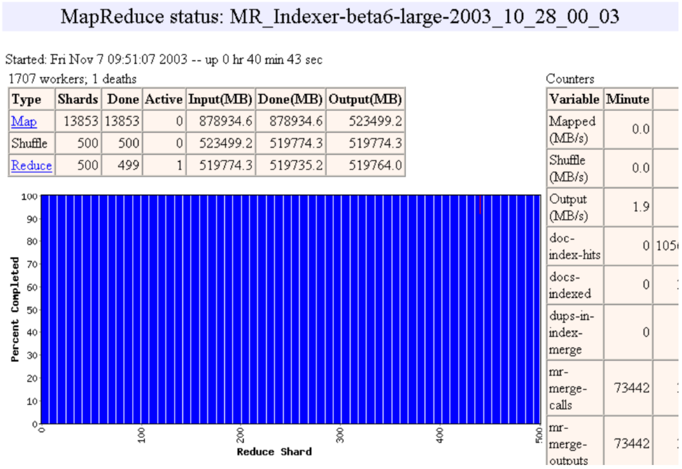
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 38 min 56 sec

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

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
- Master failure:
  - Could handle, but don't yet (master failure unlikely)

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

Semantics in presence of failures: see paper

Refinement: Redundant Execution

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 two failures for same record:
  - Next worker is told to skip the record

Effect: Can work around bugs in third-party libraries

Other Refinements (see paper)

- Sorting guarantees within each reduce partition
- Compression of intermediate data
- Combiner: useful for saving network bandwidth
- Local execution for debugging/testing
- User-defined counters

Performance

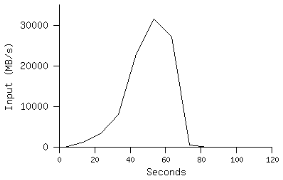
Tests run on cluster of 1800 machines:

- 4 GB of memory
- Dual-processor 2 GHz Xeons with Hyperthreading
- Dual 160 GB IDE disks
- Gigabit Ethernet per machine
- Bisection bandwidth approximately 100 Gbps

Two benchmarks:

MR_Grep	Scan 1010 100-byte records to extract records matching a rare pattern (92K matching records)
MR_Sort	Sort 1010 100-byte records (modeled after TeraSort benchmark)

MR_Grep



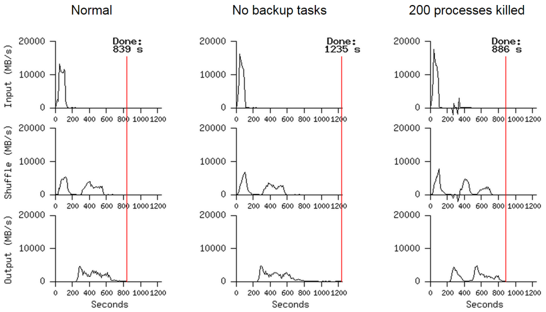
Locality optimization helps:

- 1800 machines read 1 TB of data at peak of ≈ 31 GB/s
- Without this, rack switches would limit to 10 GB/s

Startup overhead is significant for short jobs

MR_Sort

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



Experience: Rewrite of Production Indexing System

Rewrote Google's production indexing system using MapReduce

- Set of ~~10~~, ~~14~~, ~~17~~, ~~21~~, 24 MapReduce operations
- New code is simpler, easier to understand
- MapReduce takes care of failures, slow machines
- Easy to make indexing faster by adding more machines

Usage: MapReduce jobs run in August 2004

Number of jobs	29,423	
Average job completion time	634	secs
Machine days used	79,186	days
Input data read	3,288	TB
Intermediate data produced	758	TB
Output data written	193	TB
Average worker machines per job	157	
Average worker deaths per job	1.2	
Average map tasks per job	3,351	
Average reduce tasks per job	55	
Unique map implementations	395	
Unique reduce implementations	269	
Unique map/reduce combinations	426	

Related Work

- Programming model inspired by functional language primitives
- Partitioning/shuffling similar to many large-scale sorting systems
  - NOW-Sort ['97]
- Re-execution for fault tolerance
  - BAD-FS ['04] and TACC ['97]
- Locality optimization has parallels with Active Disks/Diamond work
  - Active Disks ['01], Diamond ['04]
- Backup tasks similar to Eager Scheduling in Charlotte system
  - Charlotte ['96]
- Dynamic load balancing solves similar problem as River's distributed queues
  - River ['99]

Conclusions

- MapReduce has proven to be a useful abstraction
- Greatly simplifies large-scale computations at Google
- Fun to use: focus on problem, let library deal w/ messy details

Thanks to Josh Levenberg, who has made many significant improvements and to everyone else at Google who has used and helped to improve MapReduce.