MapReduce

Simplified Data Processing on Large Clusters
(Without the Agonizing Pain)

Presented by Aaron Nathan

The Problem

- Massive amounts of data
  - >100TB (the internet)
  - Needs simple processing
- Computers aren't perfect
  - Slow
  - Unreliable
  - Misconfigured
- Requires complex (i.e. bug prone) code

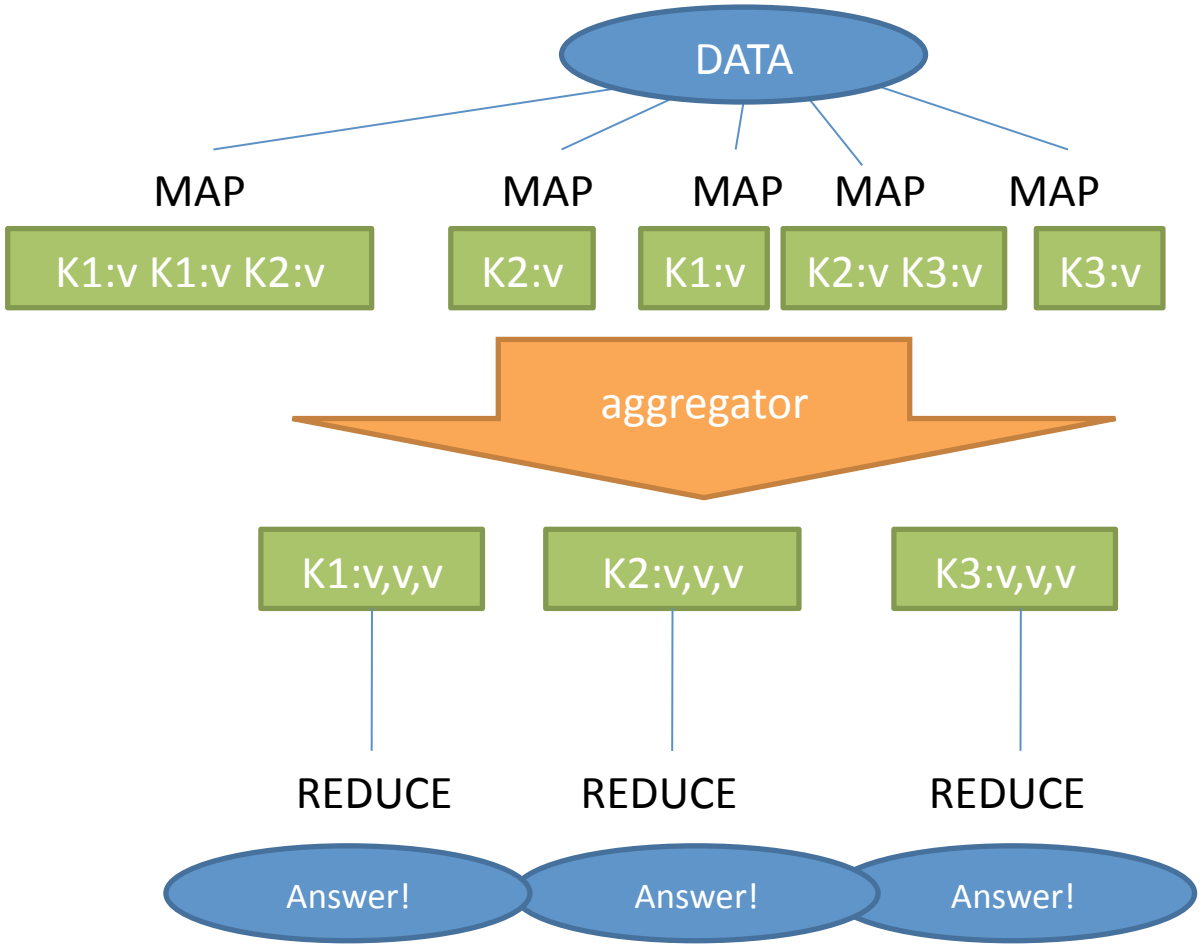
MapReduce to the Rescue!

- Common Functional Programming Model
  - Map Step

    `map (in_key, in_value) -> list(out_key, intermediate_value)`
    - Split a problem into a lot of smaller subproblems
  - Reduce Step

    `reduce (out_key, list(intermediate_value)) -> list(out_value)`
    - Combine the outputs of the subproblems to give the original problem's answer
- Each "function " is independent
- Highly Parallelizable

Algorithm Picture



Some Example Code

```
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));
```

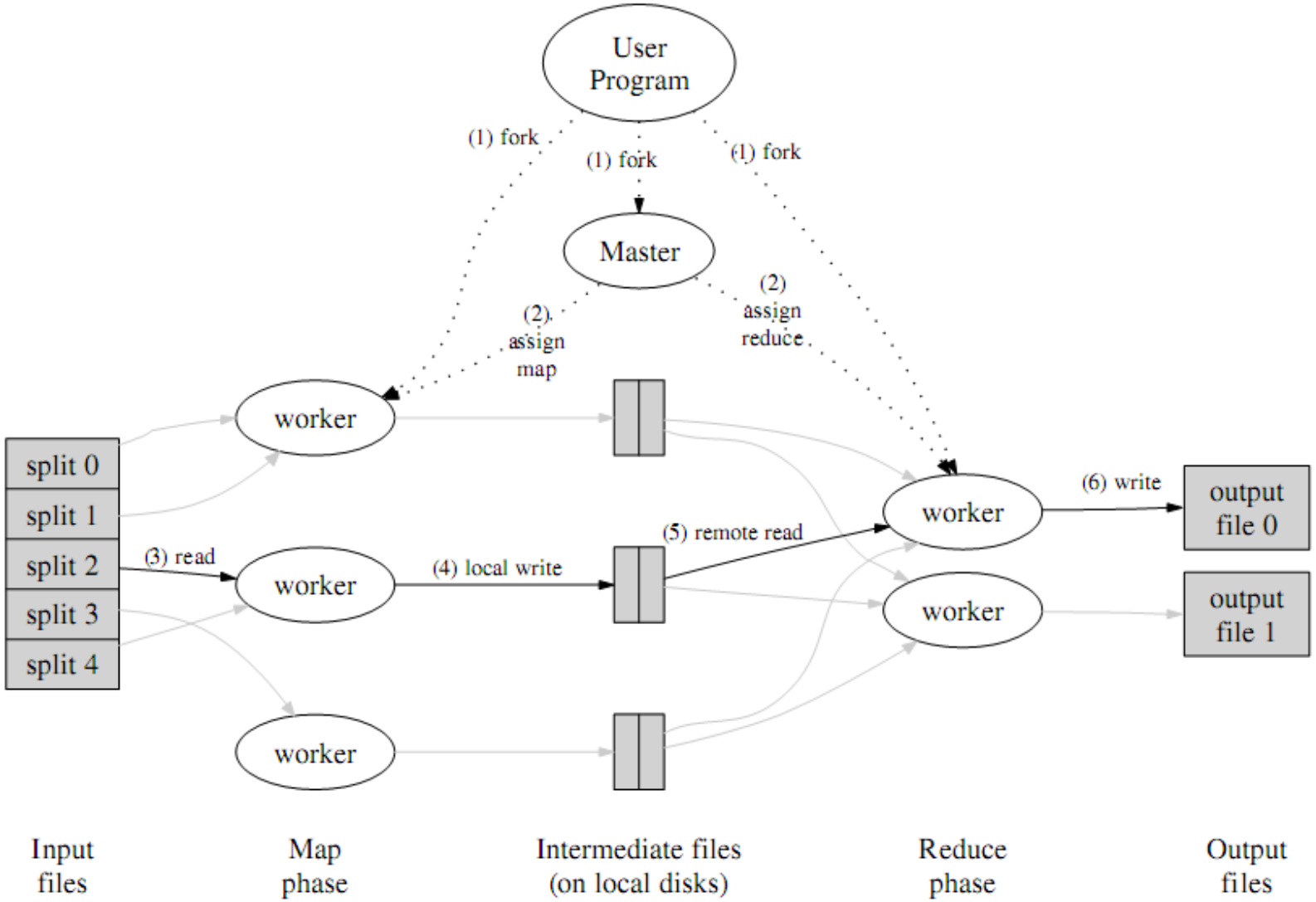
Some Example Applications

- Distributed Grep
- URL Access Frequency Counter
- Reverse Web Link Graph
- Term-Vector per Host
- Distributed Sort
- Inverted Index

The Implementation

- Google Clusters
  - 100s-1000s Dual Core x86 Commodity Machines
  - Commodity Networking (100mbps/1Gbps)
  - GFS
- Google Job Scheduler
- Library linked in c++

Execution


User Program
(1) fork
(1) fork
(1) fork
Master
(2) assign map
(2) assign reduce
worker
worker
worker
worker
worker
split 0
split 1
split 2
split 3
split 4
(3) read
(4) local write
(5) remote read
(6) write
output file 0
output file 1
Input files
Map phase
Intermediate files (on local disks)
Reduce phase
Output files

The Master

- Maintains the state and identify of all workers
- Manages intermediate values
- Receives signals from Map workers upon completion
- Broadcasts signals to Reduce workers as they work
- Can retask completed Map workers to Reduce workers.

In Case of Failure

- Periodic Pings from Master->Workers
  - On failure resets state of assigned task of dead worker
- Simple system proves resilient
  - Works in case of a 80 simultaneous machine failures!
- Master failure is unhandled.
- Worker Failure doesn't effect output
  (output identical whether failure occurs or not)
  - Each map writes to local disk only
  - If a mapper is lost, the data is just reprocessed
  - Non-deterministic map functions aren't guaranteed

Preserving Bandwidth

- Machines are in racks with small interconnects
  - Use location information from GFS
  - Attempts to put tasks for workers and input slices on the same rack
  - Usually results in LOCAL reads!

Backup Execution Tasks

- What if one machine is slow?
- Can delay the completion of the entire MR Operation!

- Answer: Backup (Redundant) Executions
  - Whoever finishes first completes the task!
  - Enabled towards the end of processing

Partitioning

- M = number of Map Tasks
  (the number of input splits)
- R = number of Reduce Tasks
  (the number of intermediate key splits)
- W = number of worker computers
- In General:
  - M = sizeof(Input)/64 MB
  - R = W*n (where n is a small number)
- Typical Scenario:
  InputSize = 12 TB,
  M = 200,000,
  R = 5000
  W = 2000

Custom Partitioning

- Default Partitioned on intermediate key
  - *Hash(intermediate_key) mod R*
- What if user has apriori knowledge about the key?
  - Allow for user-defined hashing function
  - Ex. *Hash(Hostname(url_key))*

The Combiner

- If reducer is associative and communivitive
  - (2 + 5) + 4 = 11 or 2 + (5 + 4) = 11
  - (15+x)+2=2+(15+x)
- Repeated intermediate keys can be merged
  - Saves network bandwidth
  - Essentially like a local reduce task

I/O Abstractions

- How to get initial key value pairs to map?
  - Define an input "format"
  - Make sure splits occur in reasonable places
  - Ex: Text
    - Each line is a key/pair
    - Can come from GFS, bigTable, or anywhere really!
  - Output works analogously

Skipping Bad Records

- What if a user makes a mistake in map/reduce
- And only apparent on few jobs..

- Worker sends message to Master
- Skip record on >1 worker failure and tell others to ignore this record

Removing Unnecessary Development Pain

- Local MapReduce Implementation that runs on development machine
- Master has HTTP page with status of entire operation
  - Shows “bad records”
- Provide a Counter Facility
  - Master aggregates “counts” and displayed on Master HTTP page

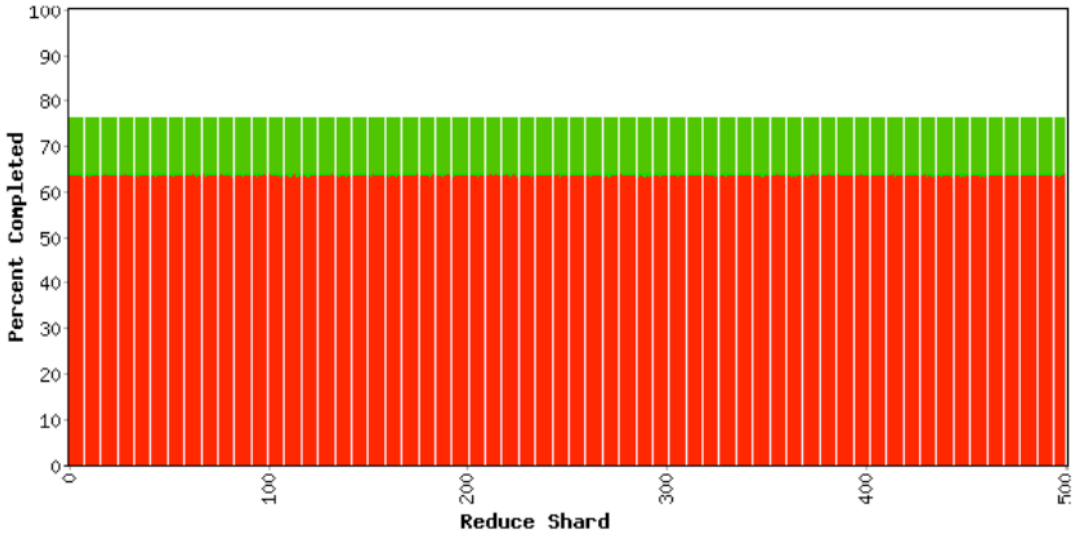
A look at the UI (in 1994)

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

http://labs.google.com/papers/mapreduce-osdi04-slides/index-auto-0013.html

Performance Benchmarks

Sorting AND Searching

Search (Grep)

- Scan through 10^{10} *100 byte* records (1TB)
- M = 15000, R = 1
- Startup time
  - GFS Localization
  - Program Propagation
- Peak -> 30 GB/sec!

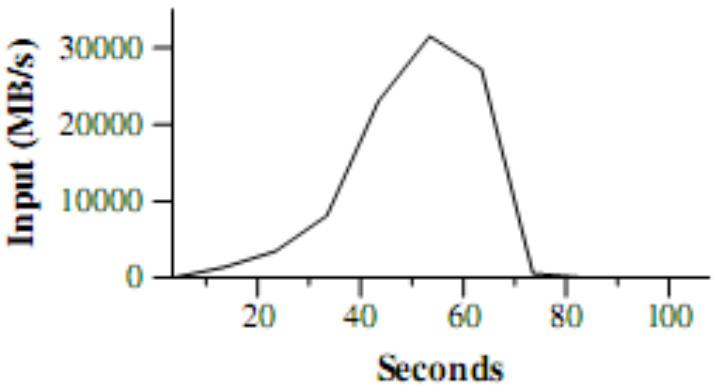
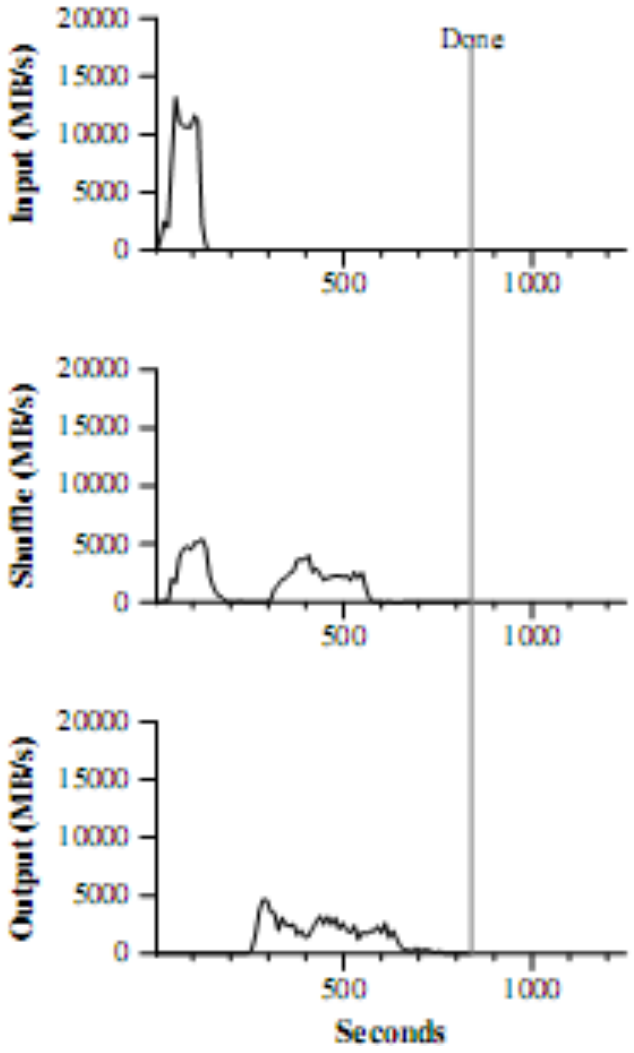


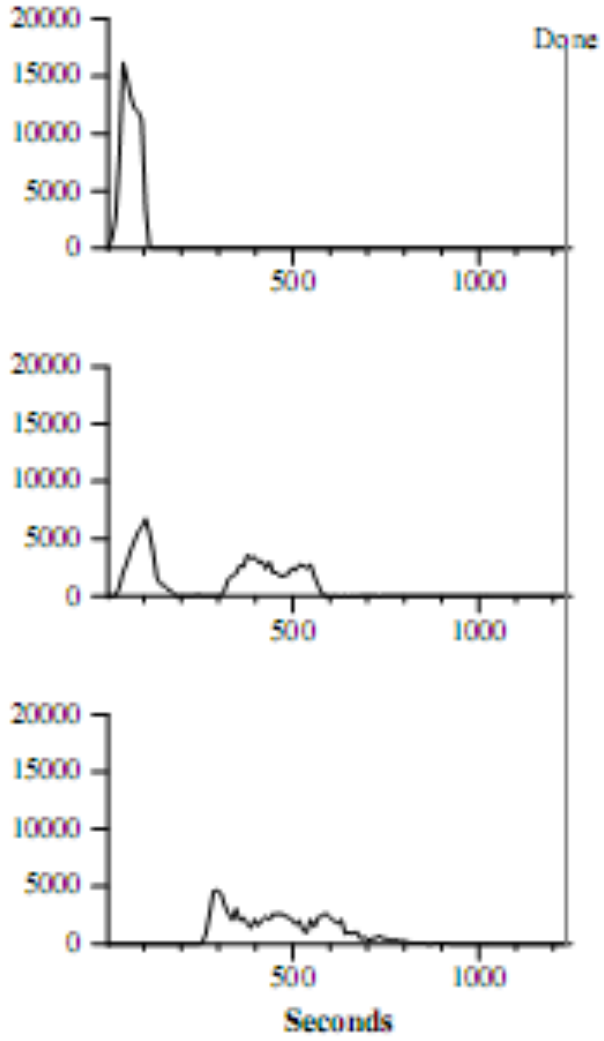
Figure 2: Data transfer rate over time

Sort

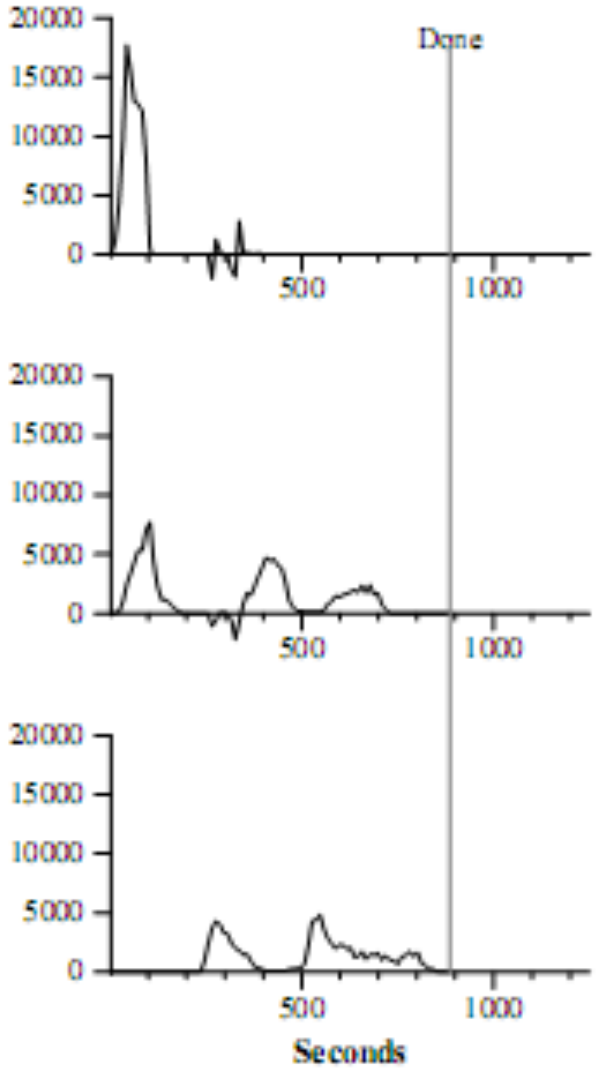
- 50 lines of code
- Map->key + textline
- Reduce-> Identity
- M = 15000, R = 4000
  - Partition on init bytes of intermediate key
- Sorts in 891 sec!



What about Backup Tasks?

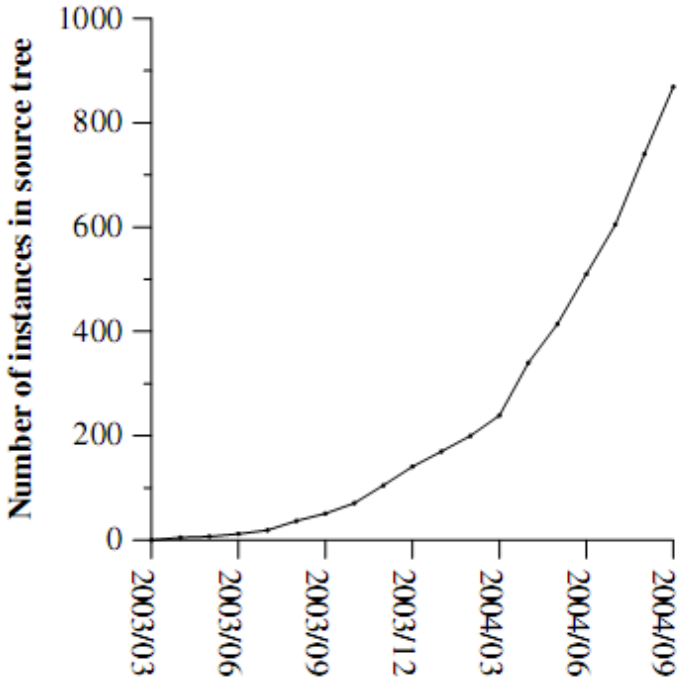


(b) No backup tasks



(c) 200 tasks killed

And wait…it's useful!



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

NB: August 2004

Open Source Implementation

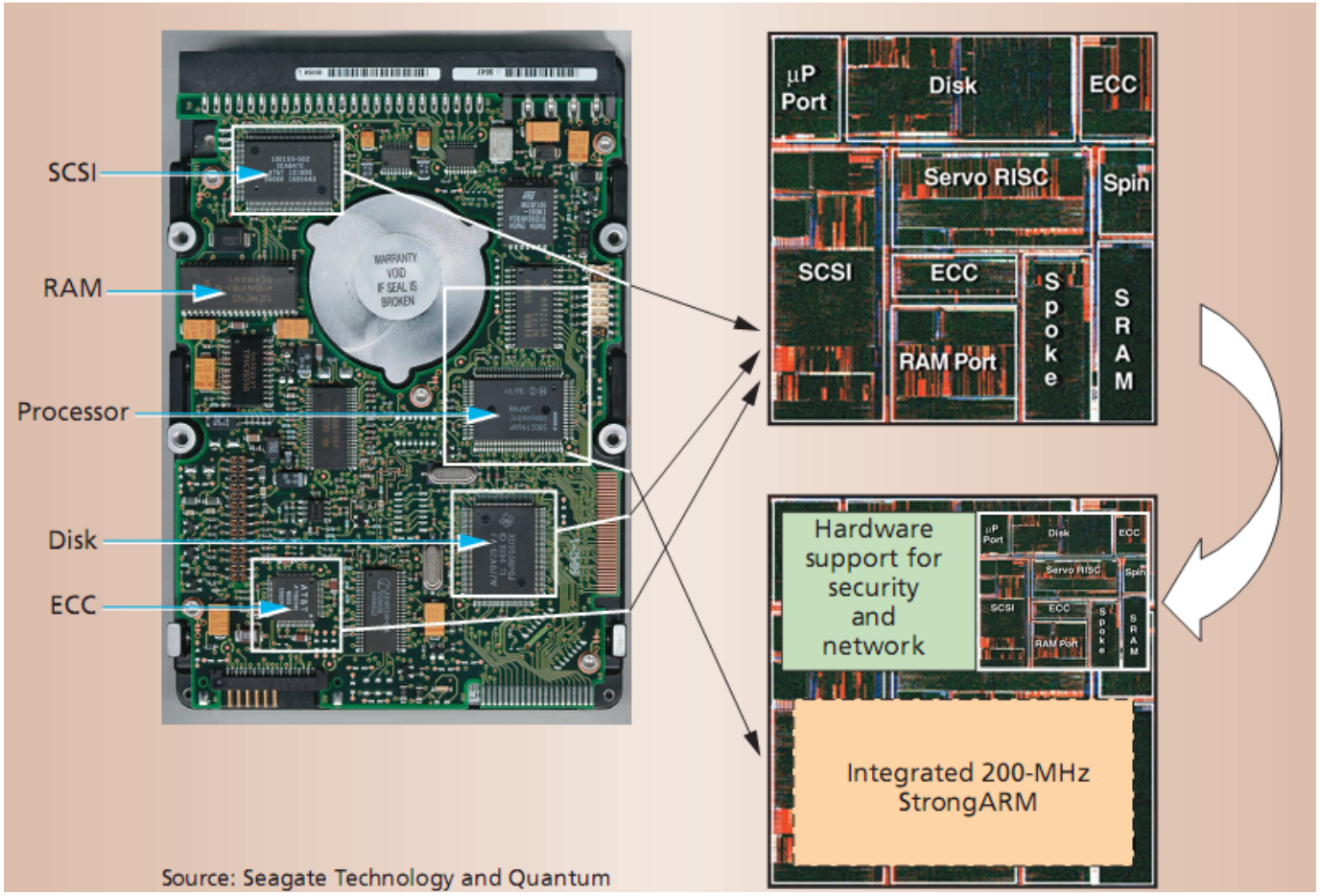
- Hadoop
- http://hadoop.apache.org/core/
  - Relies on HDFS
  - All interfaces look almost exactly like MapReduce paper
- There is even a talk about it today!
  - 4:15 B17 CS Colloquium:
    Mike Cafarella (Uwash)

Active Disks for Large-Scale Data Processing

The Concept

- Use aggregate processing power
  - Networked disks allow for higher throughput
- Why not move part of the application onto the disk device?
  - Reduce data traffic
  - Increase parallelism further

Shrinking Support Hardware…



Example Applications

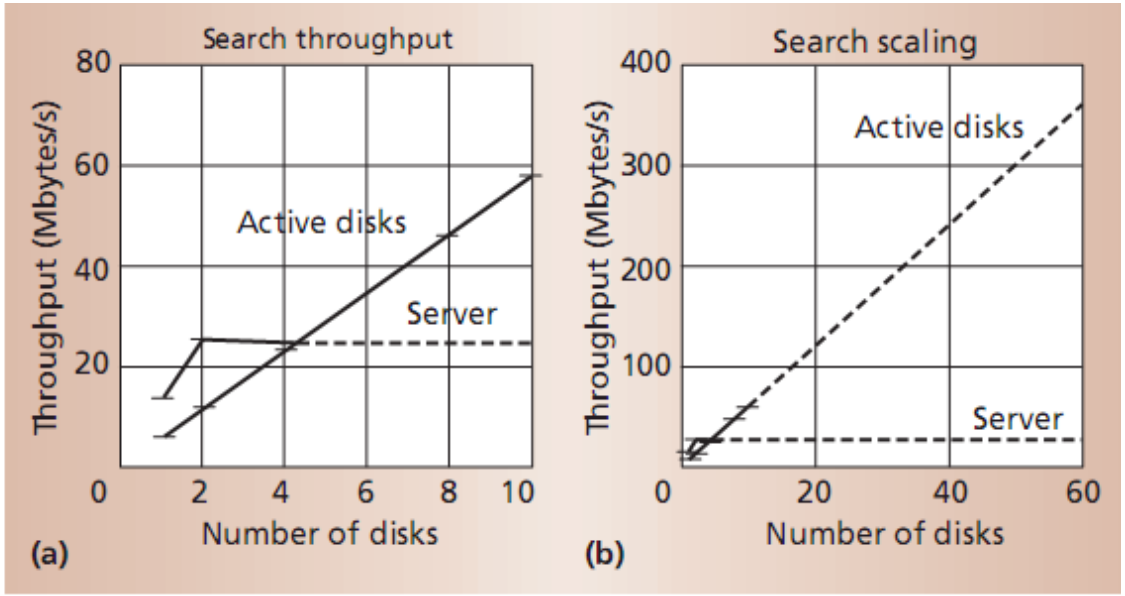
- Media Database
  - Find similar media data by “fingerprint”
- Real Time Applications
  - Collect multiple sensor data quickly
- Data Mining
  - POS Analysis required adhoc database queries

Approach

- Leverage the parallelism available in systems with many disks
- Operate with a small amount of state, processing data as it streams off the disk
- Execute relatively few instructions per byte of data

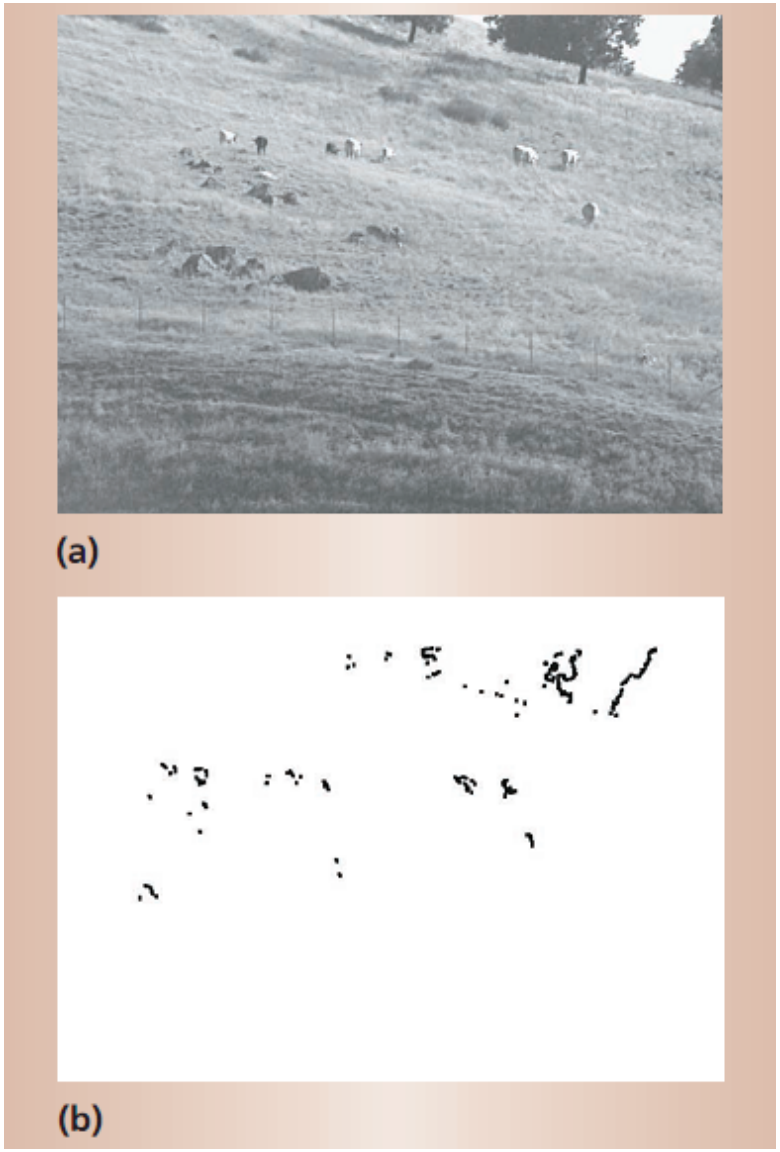
Results- Nearest Neighbor Search

- Problem: Determine k items closest to a particular item in a database
  - Perform comparisons on the drive
  - Returns the disks closest matches
  - Server does final merge



Media Mining Example

- Perform low level image tasks on the disk!
- Edge Detection performed on disk
  - Sent to server as edge image
  - Server does higher level processing



Why not just use a bunch of PC's?

- The performance incease is similar
- In fact, the paper essentially used this setup to actually benchmark their results!
- Supposedly this could be cheaper
- The paper doesn't really give a good argument for this…
  - Possibly reduced bandwidth on disk IO channel
  - But who cares?

Some Questions

- What could a disk possibly do better than the host processor?
- What added cost is associated with this mediocre processor on the HDD?
- Are new dependencies are introduced on hardware and software?
- Perhaps other (better) places to do this type of local parallel processing?
- Maybe in 2001 this made more sense?