MapReduce: Simplified Data Processing on Large Clusters

These are slides from Dan Weld's class at U. Washington
(who in turn made his slides based on those by Jeff Dean, Sanjay Ghemawat, Google, Inc.)

Motivation

- Large-Scale Data Processing
  - Want to use 1000s of CPUs
    - But don't want hassle of ***managing*** things
- MapReduce provides
  - Automatic parallelization & distribution
  - Fault tolerance
  - I/O scheduling
  - Monitoring & status updates

Map/Reduce ala Google

- map(key, val) is run on each item in set
  - emits intermediate key / val pairs

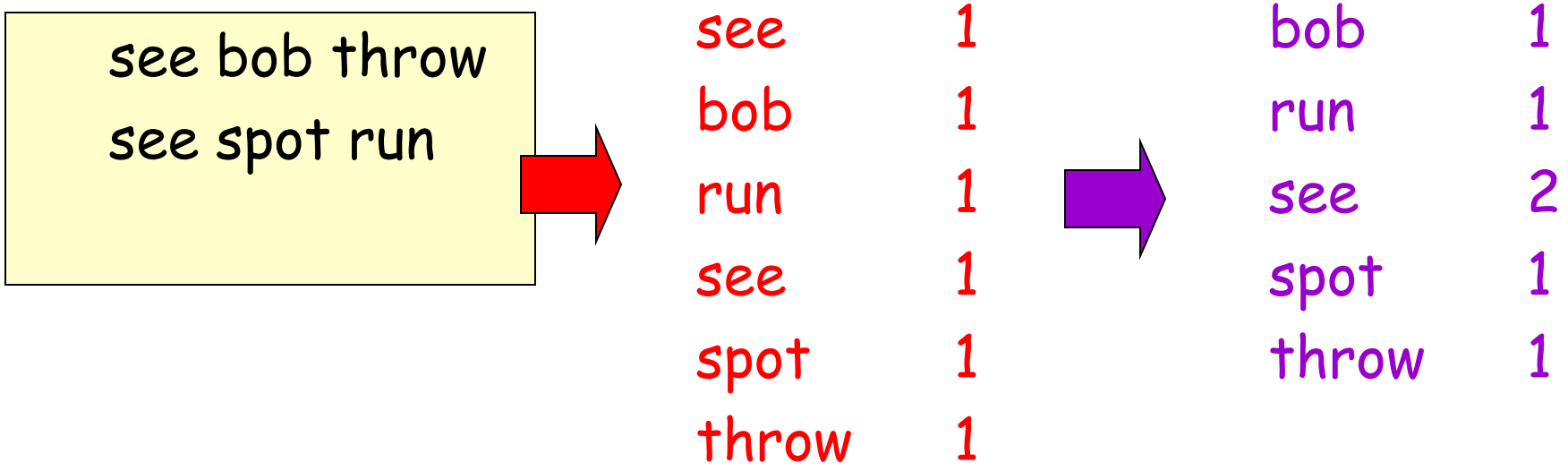
- reduce(key, vals) is run for each unique key emitted by map()
  - emits final output

count words in docs

- Input consists of (url, contents) pairs
- map(key=url, val=contents):
  - For each word *w* in contents, emit (w, "1")
- reduce(key=word, values=uniq_counts):
  - Sum all "1"s in values list
  - Emit result "(word, sum)"

Count, Illustrated

```
map(key=url, val=contents):
    For each word w in contents, emit (w, "1")
reduce(key=word, values=uniq_counts):
    Sum all "1"s in values list
    Emit result "(word, sum)"
```

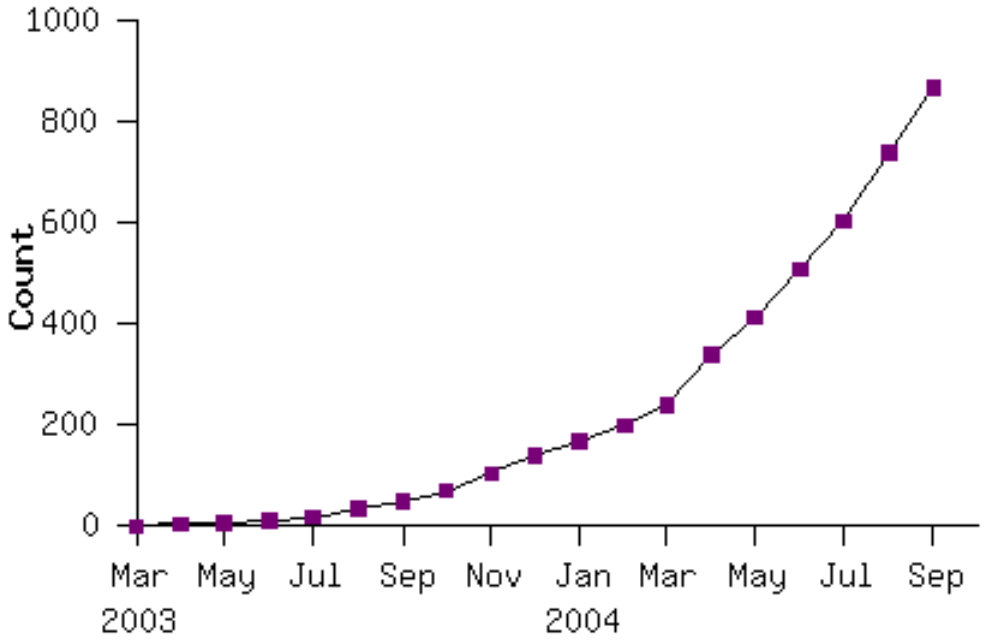


Grep

- Input consists of (url+offset, single line)
- map(key=url+offset, val=line):
  - If contents matches regexp, emit (line, "1")
- reduce(key=line, values=uniq_counts):
  - Don't do anything; just emit line

Model is Widely Applicable

MapReduce Programs In Google Source Tree



Example uses:

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

Implementation Overview

Typical cluster:

- 100s/1000s of 2-CPU x86 machines, 2-4 GB of memory
- 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

Job Processing

Fault Tolerance / Workers

Handled via re-execution

- Detect failure via periodic heartbeats
- Re-execute completed + in-progress *map* tasks
  - Why????
- Re-execute in progress *reduce* tasks
- Task completion committed through master

Robust: lost 1600/1800 machines once → finished ok

Semantics in presence of failures: see paper

Master Failure

- Could handle, … ?
- But don't yet
  - (master failure unlikely)

Refinement: Redundant Execution

Slow workers significantly delay completion time

- Other jobs consuming resources on machine
- Bad disks w/ soft errors transfer data slowly
- Weird things: processor caches disabled (!!)

Solution: Near end of phase, spawn backup tasks

- Whichever one finishes first "wins"

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

Performance

Tests run on cluster of 1800 machines:

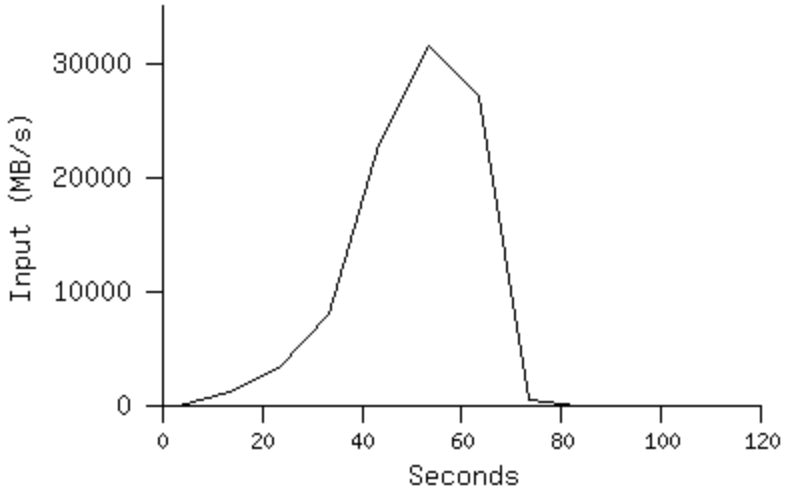
- 4 GB of memory
- Dual-processor 2 GHz Xeons with Hyperthreading
- Dual 160 GB IDE disks
- Gigabit Ethernet per machine
- Bisection bandwidth approximately 100 Gbps

Two benchmarks:

MR_GrepScan 1010 100-byte records to extract records matching a rare pattern (92K matching records)

MR_SortSort 1010 100-byte records (modeled after TeraSort benchmark)

MR_Grep



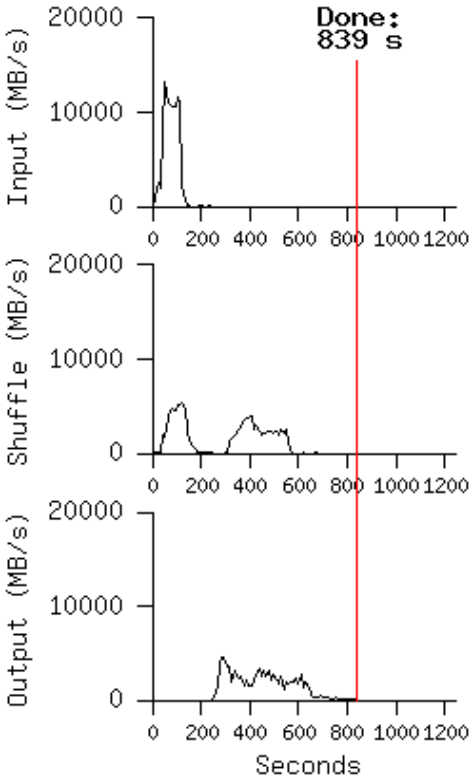
Locality optimization helps:

- 1800 machines read 1 TB at peak ~31 GB/s
- W/out this, rack switches would limit to 10 GB/s

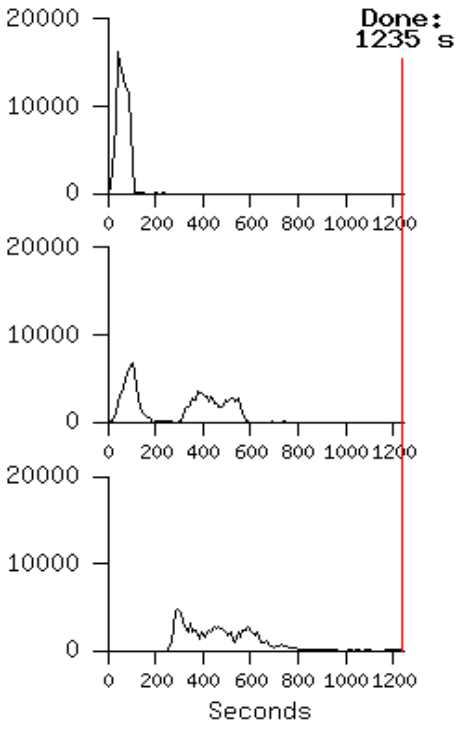
Startup overhead is significant for short jobs

MR_Sort

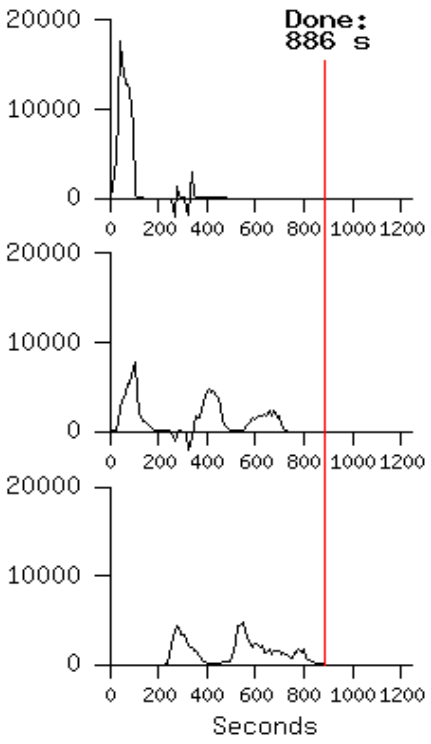
Normal



No backup tasks



200 processes killed



- Backup tasks reduce job completion time a lot!
- System deals well with failures

Conclusions

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