

Distributed computing: index building and use

1

Goals

- Do one computation faster - **latency**
- Do more computations in given time - **throughput**
- Tolerate failure of 1+ machines

2

Distributing computations

Ideas?

⇒ Finding results for a query?

- Building index?
- Goals
 - Keep all machines busy
 - Be able to replace badly-behaved machines seamlessly!

3

Distributed Query Evaluation

- Assign **different queries** to different machines
- Break up multi-term query: assign **different query terms** to different machines
 - good/bad consequences?
- Break up lexicon: assign **different index terms** to different machines?
 - good/bad consequences?
- Break up postings lists: Assign **different documents** to different machines?
 - good/bad consequences?

Keep all machines busy?

Seamlessly replace badly-behaved machines?

4

Example: Google query evaluation circa 2002

- Parallelize computation
 - distribute documents randomly to pieces of index
 - Pool of machines for each piece- choose one
 - Why random?
- Load balancing and reliability
 - Scheduler machines
 - assign tasks to pools of machines
 - monitor performance

5

Google Query Evaluation: Details circa 2002

- Enter query -> DNS-based directed to one of geographically distributed clusters
 - Load balance & fault tolerance
 - Round-trip time
- w/in cluster, query directed to 1 Google Web Server (GWS)
 - Load balance & fault tolerance
- GWS distributes query to pools of machines
 - Load sharing
- Query directed to 1 machine w/in each pool
 - Load balance & fault tolerance

6

Issues for distributed documents

- How many take from each pool to get m results?
- Throughput limits?
 - each machine does full query evaluation
 - disk access limiting constraint?
 - distributing index by term instead may help

7

Distributing computations

- ✓ Finding results for a query?
⇒ Building index?

8

Distributed Index Building

- Can easily assign different documents to different machines
- Efficient?
- Goals
 - Keep all machines busy
 - Be able to replace badly-behaved machines seamlessly!

9

Google Index Building circa 2003: MapReduce framework

- programming model
- implementation for large clusters
- Google introduced for index building and PageRank
“for processing and generating large data sets”
- The Apache Hadoop project developed open-source software
- Other applications:
 - database queries
 - join like multi-term query eval.
 - statistics on queries in given time period

10

MapReduce Programming Model

- **input set:** $\{(input\ key_i, value_i) \mid 0 \leq i \leq input\ size\}$
 - user chooses type value – e.g. whole document
- **output set:** $\{(output\ key_i, value_i) \mid 0 \leq i \leq output\ size\}$
- **Map (written by user):**
 $(input\ key, value) \rightarrow \{(intermed.\ key_j, value_j) \mid 0 \leq j \leq Map\ result\ size\}$
- **system** groups all Map output pairs for input set by **intermediate key (shuffle phase)**
 - gathers by intermediate key value
 - supply to Reduce by iterator
- **Reduce (written by user)** process intermediate values:
 $(intermed.\ key, list\ of\ values) \rightarrow (output\ key, value)$

11

MapReduce for building inverted index

- Input pair: (docID, contents of doc)
- Map: produce $\{(term, docID)\}$ for each term appearing in docID
- Input to Reduce: (term, docIDs) pairs for each term
- Output of Reduce: (term, sorted list of docIDs containing that term)
 - postings list!

keys 12

Diagram of computation distribution

See Figure 2.3 (pg 27) in
Mining of Massive Data Sets by Rajaraman,
Leskovec and Ullman

Originally appeared as Figure 1 in
*MapReduce: Simplified Data Processing on Large
Clusters* by J. Dean and S. Ghemawat,
Comm. of the ACM, vol. 51, no. 1 (2008), pp. 107-113.

13

MapReduce parallelism

- Map phase and shuffle phase may overlap
- Shuffle phase and reduce phase may overlap
- Map phase must finish before reduce phase starts
 - reduce depends on all values associated with a given key

14

MapReduce Fault Tolerance

- Master fails => restart whole computation
- Worker node fails
 - Master detects failure
 - must redo all Map tasks assigned to worker
 - output of completed Map tasks on failed worker's disk
 - for failed Map worker, Master
 - reschedules each Map task
 - notifies reducer workers of change in input location
 - for failed Reduce worker, Master
 - reschedules each Reduce task
 - rescheduling occurs as live workers become available

15

Remarks

- Google built on **large collections** of inexpensive "commodity PCs"
 - always some not functioning
- **Solve fault-tolerance problem in software**
 - redundancy & flexibility NOT special-purpose hardware
- Keep **machines** relative **generalists**
 - machine becomes free => assign to any one of set of tasks

16

June 2010 New Google index building: Caffeine

- daily crawl "several billion" documents
- Before:
 - Rebuild index: new + existing
 - series of 100 MapReduces to build index
 - "each doc. spent 2-3 days being indexed"
- After:
 - Each document fed through Percolator:
 - **incremental** update of index
 - Document indexed 100 times faster (median)
 - Avg. age doc. in search result decr. "nearly 50%"

17

Percolator

- Built on top of **Bigtable** distributed storage
 - "tens of petabytes" in indexing system
- Provides random access
 - Requires extra resources over MapReduce
- Provides **transaction** semantics
 - Repository transformation highly **concurrent**
 - Requires some **consistency** guarantees for data
- "Observers" do tasks; write to table
- Writing to table creates work for other observers
- "around 50" Bigtable op.s to process 1 doc.

18

Bigtable Overview

- Distributed database system
 - One **big** table
 - Sparse
- cells indexed by row key, column key, timestamp
 - Sorted by row key
- rows have variable number of columns
- Atomic read-modify-write by row
- Data in cell “uninterpreted strings”
 - User provide interpretation

19

Bigtable Overview: Distribution

- Rows partitioned into tablets
 - contiguous key space
- tablet servers execute operations
- Performance
 - **large** number tablet servers
- Fault tolerance
 - replication of data
 - transaction log
 - server take over for failed server

20

Percolator builds on Bigtable

- Percolator metadata stored alongside data in special columns of Bigtable
- Percolator adds functionality:
 - Multi-row transactions
 - “observer” framework

21

Percolator transactions

- maintains locks
- multiple versions each data item
 - timestamps
 - stable “snapshots” for reads
- compare database system
 - Percolator not require “extremely low latency”
 - affects approach

22

Percolator observers

- users write observer code
- run distributed across collection of machines
- observer “registers” function and set of columns with Percolator
- Percolator invokes function after data written in one of columns (any row)
 - Percolator must find “dirty” cells
 - search distributed across machines
 - avoid >1 observer for a single column

23

Caffeine versus MapReduce

- Caffeine uses “roughly twice as many resources” to process same crawl rate
- New document collection “currently 3x larger than previous systems”
 - Only limit available disk space
- Document indexed 100 times faster (median)
- If number newly-crawled docs near size index, MapReduce better
 - random lookup v.s. streaming

24