Distributed computing: index building and use

Goals

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

Distributing computations

Ideas?

⇒ Finding results for a query?

- Building index?

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

Distributed Query Evaluation

- Assign different queries to different machines
- 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?
- Goals
  - Keep all machines busy
  - Be able to replace badly-behaved machines seamlessly!

Google query evaluation
circa 2002

- Parallelize computation
  - distribute documents randomly to pieces of index
    - Pool of machines for each - choose one
    - Why random?

- Load balancing and reliability
  - Scheduler machines
    - assign tasks to pools of machines
    - monitor performance

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

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

Ideas?

✓ 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
  - programming model
  - implementation for large clusters

"for processing and generating large data sets"

- Example applications
  - ✳inverted index
  - graph structure of Web docs.
  - statistics on queries in given time period

10

MapReduce Programming Model

- input set: {(input key_i, $value_i$)| $0 \le i \le$ input size}
- output set: {(output key_i, $value_i$)| $0 \le i \le$ output size}
- Map: (input key, value) →
  {(intermediate key_j, $value_j$)| $0 \le j \le$ Map result size}
  - written by user
- 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: (intermediate key, list of values) →
  (output key*, {result values})
  - written by user to process intermediate values

* output key often constrained to be intermediate key

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: list of all (term, docID) pairs for one term
- Output of Reduce: (term, sorted list of docIDs containing that term)
  - postings list!

keys

12

Diagram of computation distribution

See Figure 1 in

MapReduce:
Simplified Data Processing on Large Clusters
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

Hadoop

"The Apache Hadoop project develops open-source software for reliable, scalable, distributed computing. "

Includes MapReduce

http://hadoop.apache.org/index.html

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 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

- Multidimensional sorted map
  - Sparse
  - Distributed
- Table partitioned into tablets
  - contiguous key space
  - tablet servers
- cells indexed by row key, column key, timestamp
  - Sorted by row key
- Data in cell “uninterpreted strings”
  - User provide interpretation
  - Supports semi-structured data
- Atomic read-modify-write by row

19

Percolator builds on Bigtable

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

20

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

21

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
- compare database “triggers”

22

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)

23