Introduction to Data Management
CSE 344

Lecture 22: MapReduce

Where We Are

• We are talking about parallel query processing

• There exist two main types of engines:
  – Parallel DBMSs (last lecture)
  – MapReduce and similar systems (this lecture)

Review: Parallel DBMS

From: Greenplum Database Whitepaper

Parallel Join: \( R \bowtie_{X=X} S \)

Typically \( R \) and \( S \) are both spread across ALL cluster nodes (machines)

Parallel Group By: \( \sum_{A} \text{sum}(B)(R) \)

Parallel Dupe-elim: \( \delta (R) \)
Parallel DBMS

- Parallel query plan: tree of parallel operators
  - Data streams from one operator to the next
  - Typically all cluster nodes process all operators
- Can run multiple queries at the same time
  - Queries will share the nodes in the cluster
- Notice that user does not need to know how his/her SQL query was processed

MapReduce

Google: [Dean 2004]
- Open source implementation: Hadoop
- MapReduce = high-level programming model and implementation for large-scale parallel data processing

Map Reduce

- Not designed to be a DBMS
- Designed to simplify task of writing parallel programs
  - A simple programming model that applies to many large-scale computing problems
- Hides messy details in MapReduce runtime library:
  - Automatic parallelization
  - Load balancing
  - Network and disk transfer optimizations
  - Handling of machine failures
  - Robustness
  - Improvements to core library benefit all users of library!

Typical Problems Solved by MR

- Read a lot of data
- Map: extract something you care about from each record
- Shuffle and Sort
- Reduce: aggregate, summarize, filter, or transform
- Write the results
  Outline stays the same, map and reduce change to fit the problem

Observation: Your favorite parallel algorithm…
MapReduce 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)

Example: What does this do?

```java
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: word
// output_values: ????
int result = 0;
for each v in intermediate_values:
  result += v;
Emit(result);
```

Parallel DBMS vs MapReduce

- Parallel DBMS
  - Relational data model and schema
  - Declarative query language: SQL
  - Many pre-defined operators: relational algebra
  - Can easily combine operators into complex queries
  - Query optimization
  - Can do more than just run queries: Data management
    - Updates and transactions, constraints, security, etc.
- MapReduce
  - Data model is a file with key-value pairs!
  - No need to "load data" before processing it
  - Easy to write user-defined operators

Implementation

- There is one master node
- Input file is split into M blocks
- Blocks are stored on random machines and replicated
- Master partitions input file further into $M'$ splits, by key
- Master assigns workers (=servers) to the $M'$ map tasks, keeps track of their progress
- Workers write their output to local disk
- Output of each map task is partitioned into $R$ regions
- Master assigns workers to the $R$ reduce tasks
- Reduce workers read regions from the map workers' local disks

MR Phases
Interesting Implementation Details

• Worker failure:
  – Master pings workers periodically,
  – If down then reassigns its split to another worker
  – (# a parallel DBMS restarts whole query)

• How many map and reduce tasks:
  – Larger is better for load balancing
  – But more tasks also add overheads
  – (# parallel DBMS spreads ops across all nodes)

Backup tasks:

  • Straggler = a machine that takes unusually long
time to complete one of the last tasks. Eg:
  – Bad disk forces frequent correctable errors (30MB/s →
    1MB/s)
  – The cluster scheduler has scheduled other tasks on
    that machine
  • Stragglers are a main reason for slowdown
  • Solution: pre-emptive backup execution of the
    last few remaining in-progress tasks

Parallel DBMS vs MapReduce

• Parallel DBMS
  – Indexing
  – Physical tuning
  – Can stream data from one operator to the next without blocking

• MapReduce
  – Can easily add nodes to the cluster (no need to even restart)
  – Uses less memory since processes on key-group at a time
  – Intra-query fault-tolerance thanks to results on disk
  – Intermediate results on disk also facilitate scheduling
  – Handles adverse conditions: e.g., stragglers
  – Arguably more scalable… but also need more nodes!

MapReduce State

• Lots of extensions to address limitations
  – Capabilities to write DAGs of MapReduce jobs
  – Declarative languages (next lecture)
  – Ability to read from structured storage (e.g., indexes)
  – Etc.

• Most companies use both types of engines
• Increased integration of both engines
• Competing engine: Dryad from MS

HW6

• We will use MapReduce (Hadoop)
• We will use a declarative language: Pig Latin
• Cluster will run in Amazon’s cloud
  – Give your credit card
  – Click, click, click… and you have a MapReduce
    cluster running all configured for you
• We will analyze a real 0.5TB graph