Data-Intensive Distributed Computing
CS 431/631 451/651 (Fall 2019)

Part 1: MapReduce Algorithm Design (3/4)

Ali Abedi

These slides are available at https://www.student.cs.uwaterloo.ca/~cs451/

This work is licensed under a Creative Commons Attribution-Noncommercial-Share Alike 3.0 United States
See http://creativecommons.org/licenses/by-nc-sa/3.0/us/ for details
Agenda for Today

- Cloud computing
- Datacenter architectures
- Hadoop cluster architecture
- MapReduce physical execution
Today

- Data Science Tools
- Analytics Infrastructure
- Execution Infrastructure

"big data stack"
Aside: Cloud Computing
The best thing since sliced bread?

Before clouds...

Grids
supercomputers

Cloud computing means many different things:

Big data
Rebranding of web 2.0
Utility computing
Everything as a service
Rebranding of web 2.0

Rich, interactive web applications
Clouds refer to the servers that run them
Examples: Facebook, YouTube, Gmail, ...

“The network is the computer”: take two
User data is stored “in the clouds”
Rise of the tablets, smartphones, etc. (“thin clients”)
Browser is the OS
Utility Computing

What?
Computing resources as a metered service ("pay as you go")

Why?
Cost: capital vs. operating expenses
Scalability: "infinite" capacity
Elasticity: scale up or down on demand

Does it make sense?
Benefits to cloud users
Business case for cloud providers

I think there is a world market for about five computers.
Evolution of the Stack

Traditional Stack

Virtualized Stack

Containerized Stack
Everything as a Service

Infrastructure as a Service (IaaS)
Why buy machines when you can rent them instead?
Examples: Amazon EC2, Microsoft Azure, Google Compute

Platform as a Service (PaaS)
Give me a nice platform and take care of maintenance, upgrades, ...
Example: Google App Engine

Software as a Service (SaaS)
Just run the application for me!
Example: Gmail, Salesforce
Everything as a Service

Database as a Service
Run a database for me
Examples: Amazon RDS, Microsoft Azure SQL, Google Cloud BigTable

Search as a Service
Run a search engine for me
Example: Amazon Elasticsearch Service

Function as a Service
Run this function for me
Example: Amazon Lambda, Google Cloud Functions
Who cares?

A source of problems...
Cloud-based services generate big data
Clouds make it easier to start companies that generate big data

As well as a solution...
Ability to provision clusters on-demand in the cloud
Commoditization and democratization of big data capabilities
So, what *is* the cloud?
What is the Matrix?
Building Blocks

Source: Barroso and Urs Hölzle (2009)
Anatomy of a Datacenter

Source: Barroso and Urs Hölzle (2013)
Datacenter cooling

Source: Barroso and Urs Hölzle (2013)
How much is 30 MW?
Datacenter Organization

Source: Barroso and Urs Hölzle (2013)
The datacenter is the computer!

It’s all about the right level of abstraction
Moving beyond the von Neumann architecture
What’s the “instruction set” of the datacenter computer?

Hide system-level details from the developers
No more race conditions, lock contention, etc.
No need to explicitly worry about reliability, fault tolerance, etc.

Separating the what from the how
Developer specifies the computation that needs to be performed
Execution framework (“runtime”) handles actual execution

Wait, why do we care?
Mechanical Sympathy

“You don’t have to be an engineer to be a racing driver, but you do have to have mechanical sympathy”
– Formula One driver Jackie Stewart

“big data stack”
Intuitions of time and space

How long does it take to read 100 TBs from 100 hard drives?
Now, what about SSDs?

How long will it take to exchange 1b key-value pairs:
Between machines on the same rack?
Between datacenters across the Atlantic?
Storage Hierarchy

Local Machine
- L1/L2/L3 cache, memory, SSD, magnetic disks
- capacity, latency, bandwidth

Remote Machine
- Same Rack
- Different Rack
- Different Datacenter
<table>
<thead>
<tr>
<th>Operation</th>
<th>Time (ns)</th>
</tr>
</thead>
<tbody>
<tr>
<td>L1 cache reference</td>
<td>0.5</td>
</tr>
<tr>
<td>Branch mispredict</td>
<td>5</td>
</tr>
<tr>
<td>L2 cache reference</td>
<td>7</td>
</tr>
<tr>
<td>Mutex lock/unlock</td>
<td>100</td>
</tr>
<tr>
<td>Main memory reference</td>
<td>100</td>
</tr>
<tr>
<td>Compress 1K bytes with Zippy</td>
<td>10,000</td>
</tr>
<tr>
<td>Send 2K bytes over 1 Gbps network</td>
<td>20,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from memory</td>
<td>250,000</td>
</tr>
<tr>
<td>Round trip within same datacenter</td>
<td>500,000</td>
</tr>
<tr>
<td>Disk seek</td>
<td>10,000,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from network</td>
<td>10,000,000</td>
</tr>
<tr>
<td>Read 1 MB sequentially from disk</td>
<td>30,000,000</td>
</tr>
<tr>
<td>Send packet CA-&gt;Netherlands-&gt;CA</td>
<td>150,000,000</td>
</tr>
</tbody>
</table>
Hadoop Cluster Architecture
How do we get data to the workers?
Let’s consider a typical supercomputer...
Sequoia will enable simulations that explore phenomena at a level of detail never before possible. Sequoia is dedicated to NNSA’s Advanced Simulation and Computing (ASC) program for stewardship of the nation’s nuclear weapons stockpile, a joint effort from LLNL, Los Alamos National Laboratory and Sandia National Laboratories.
Compute-Intensive vs. Data-Intensive

Why does this make sense for compute-intensive tasks? What’s the issue for data-intensive tasks?
What’s the solution?
Don’t move data to workers... move workers to the data!

Key idea: co-locate storage and compute
Start up worker on nodes that hold the data
What’s the solution?
Don’t move data to workers... move workers to the data!

Key idea: co-locate storage and compute

Start up worker on nodes that hold the data

We need a distributed file system for managing this

GFS (Google File System) for Google’s MapReduce
HDFS (Hadoop Distributed File System) for Hadoop
GFS: Assumptions

Commodity hardware over “exotic” hardware
  Scale “out”, not “up”

High component failure rates
  Inexpensive commodity components fail all the time

“Modest” number of huge files
  Multi-gigabyte files are common, if not encouraged

Files are write-once, mostly appended to
  Logs are a common case

Large streaming reads over random access
  Design for high sustained throughput over low latency

GFS slides adapted from material by (Ghemawat et al., SOSP 2003)
GFS: Design Decisions

Files stored as chunks
  Fixed size (64MB)

Reliability through replication
  Each chunk replicated across 3+ chunkservers

Single master to coordinate access and hold metadata
  Simple centralized management

No data caching
  Little benefit for streaming reads over large datasets

Simplify the API: not POSIX!
  Push many issues onto the client (e.g., data layout)

HDFS = GFS clone (same basic ideas)
From GFS to HDFS

Terminology differences:
GFS master = Hadoop namenode
GFS chunkservers = Hadoop datanodes

Implementation differences:
Different consistency model for file append
Implementation language
Performance

For the most part, we’ll use Hadoop terminology...
HDFS Architecture

Adapted from (Ghemawat et al., SOSP 2003)
Namenode Responsibilities

Managing the file system namespace
Holds file/directory structure, file-to-block mapping, metadata (ownership, access permissions, etc.)

Coordinating file operations
Directs clients to datanodes for reads and writes
No data is moved through the namenode

Maintaining overall health
Periodic communication with the datanodes
Block re-replication and rebalancing
Garbage collection
Logical View

\[
\begin{align*}
\text{k}_1 & \quad \text{v}_1 \\
\text{k}_2 & \quad \text{v}_2 \\
\text{k}_3 & \quad \text{v}_3 \\
\text{k}_4 & \quad \text{v}_4 \\
\text{k}_5 & \quad \text{v}_5 \\
\text{k}_6 & \quad \text{v}_6
\end{align*}
\]

\[
\begin{align*}
\text{map} & \quad \text{map} & \quad \text{map} & \quad \text{map} \\
\text{a} & \quad \text{b} & \quad \text{c} & \quad \text{c} & \quad \text{a} & \quad \text{b} & \quad \text{c} & \quad \text{a} & \quad \text{b} & \quad \text{c} & \quad \text{a} & \quad \text{b} & \quad \text{c}
\end{align*}
\]

\[
\begin{align*}
\text{combine} & \quad \text{combine} & \quad \text{combine} & \quad \text{combine} \\
\text{a} & \quad \text{b} & \quad \text{c} & \quad \text{c} & \quad \text{a} & \quad \text{b} & \quad \text{c} & \quad \text{a} & \quad \text{b} & \quad \text{c} & \quad \text{a} & \quad \text{b} & \quad \text{c}
\end{align*}
\]

\[
\begin{align*}
\text{partition} & \quad \text{partition} & \quad \text{partition} & \quad \text{partition} \\
\text{a} & \quad \text{b} & \quad \text{c} & \quad \text{c} & \quad \text{a} & \quad \text{b} & \quad \text{c} & \quad \text{a} & \quad \text{b} & \quad \text{c} & \quad \text{a} & \quad \text{b} & \quad \text{c}
\end{align*}
\]

\[
\text{group values by key}
\]

\[
\begin{align*}
\text{a} & \quad \text{b} & \quad \text{c} \\
\text{1} & \quad \text{5} & \quad \text{2} & \quad \text{7} & \quad \text{2} & \quad \text{9} & \quad \text{8}
\end{align*}
\]

\[
\begin{align*}
\text{reduce} & \quad \text{reduce} & \quad \text{reduce} \\
\text{r}_1 & \quad \text{s}_1 & \quad \text{r}_2 & \quad \text{s}_2 & \quad \text{r}_3 & \quad \text{s}_3
\end{align*}
\]
Physical View

Adapted from (Dean and Ghemawat, OSDI 2004)
Adapted from (Ghemawat et al., SOSP 2003)
Putting everything together...
Basic Cluster Components*

Namenode (NN)
Master for HDFS

Jobtracker (JT)
Coordinator for MapReduce jobs

On each of the worker machines:
Tasktracker (TT): contains multiple task slots
Datanode (DN): serves HDFS data blocks

* Not quite... leaving aside YARN for now
What are these input split?
What are these input split?
Mapper

Intermediates

Partitioner

Reducer

What's going on here?

(combiners omitted here)
Distributed Group By in MapReduce

Map side
Map outputs are buffered in memory in a circular buffer
When buffer reaches threshold, contents are “spilled” to disk
Spills are merged into a single, partitioned file (sorted within each partition)
Combiner runs during the merges

Reduce side
First, map outputs are copied over to reducer machine
“Sort” is a multi-pass merge of map outputs (happens in memory and on disk)
Combiner runs during the merges
Final merge pass goes directly into reducer
Distributed Group By in MapReduce

Mapper

- circular buffer (in memory)
- spills (on disk)
- other mappers

Reducer

- Combiner
- intermediate files (on disk)
- other reducers

But runtime can begin copying intermediate data earlier
group values by key

Why?

* Important detail: reducers process keys in sorted order
Law of Leaky Abstractions

All non-trivial abstractions, to some degree, are leaky.

Joel Spolsky

Remember logical vs. physical?