Big Data Technology
Core Hadoop: HDFS-YARN and MR Internals

Eshcar Hillel
Yahoo!

Ronny Lempel
Outbrain
Roadmap

- Previous class – Map-Reduce Motivation

- This class – HDFS-YARN and MR overview
  - HDFS - Distributed Filesystem
  - YARN - Scheduling and Resource Management (Yet Another Resource Negotiator)
  - MR Runtime in a Nutshell
  - MR Performance Optimizations
Percolator: Incremental Processing System

HDFS-YARN: The core of Hadoop

Source: http://hortonworks.com/hdp/
Hardware Layout

1000s nodes

Few dozens

Base Switch

Rack Switch

Rack

Node
Example: Yahoo Hadoop Cluster
Core Hadoop: The Big Picture

Source: http://www.slideshare.net/Hadoop_Summit/hadoop-crash-course-workshop-at-hadoop-summit
HDFS 101

- Highly available cost-efficient distributed storage system
- Highly scalable
  - ~5 thousand commodity servers clusters
  - Up to 200 PB data
  - Billions files and block
- Master/Worker architecture
How HDFS Works?

- Files are write-once-read-many
  - Optimized for appends and sequential scans
- Data replication
  - Files split into large blocks (128MB) (why so large?)
  - Each block is replicated at multiple nodes
  - Typically, 3 replicas per block, might be more
- Placement policies
  - Typically, 2 replicas in a single rack, 1 replica in a remote rack (why?)
NameNode (NN) DataNodes (DNs)

- **NameNode** manages cluster metadata
  - Namespace tree
  - Blocks to datanodes mapping

- **DataNodes** store data
  - Blocks are stored on local filesystem
  - Send heartbeats to NameNode
  - Get instructions in response
    - Remove/replicate block, shutdown, block report
Data Read and Write Operations

Data servers (DataNodes)

Client

Metadata access

File

Metadata server (NameNode)
Data Read and Write Operations

- Each replica can serve the data
  - Served from a replica that is closest to the reader

- Single-writer semantics
  - Obtain lease for writing
  - Per-block: get datanode list to host replication
  - Per-block: replication pipeline
Replication Pipeline

Source: http://blog.cloudera.com/blog/2015/02/understanding-hdfs-recovery-processes-part-1/

CS236620 Big Data Technology
HDFS Scalability

- The scale of data
  - Past (2007): 100s nodes, 100s TB, millions files
  - Present (2015): 1000s nodes (10x), 200PB (1000x), 100s millions files (100x)
  - Future: 10000s, xxxEB, Billions of files

- Namenode limits the cluster scale
  - Does not scale beyond the size of the NN heap

- (Future) Extend scalability by decoupling namespace and block manager
  - Namespace scalability via key-value stores (HDFS-8286)
  - Block manager amenable to sharding/scale-out (HDFS-5477)
Highly Available NameNode

- Backup NameNode provides redundancy and supports high availability (HA)

```
Highly Available NameNode

Backup NameNode provides redundancy and supports high availability (HA)

FailoverController
ZK
ZK
ZK

Journal Nodes

Metadata (backup NN)
Metadata (primary NN)

Monitor health
Monitor health
Monitor health

heartbeat
heartbeat

Data service (datanode)

CS236620 Big Data Technology
```
YARN 101

- A framework for job scheduling and resource management
  - Distributes computation across HDFS nodes
  - Enables a variety of concurrent data access applications
- Master/worker architecture
YARN Architecture

Resource Manager

Node Manager
- AppMaster B
- AppMaster A
- Job A M3
- Job A R1
- Job B M2
- Job B R2

Node Manager
- Job A M4
- Job A R1
- Job B M2
- Job B R1

Node Manager
- Job C M1
- Job C R1
- Job B M1
- Job A M1

Containers

CS236620 Big Data Technology
YARN Architecture

- **Central ResourceManager (RM)**
  - Constraints-based scheduler allocating resources
    - Capacity, queues, etc.
  - No monitoring or status tracking

- **Per-Node NodeManager (NM)**
  - Manages available resources in a single node
  - Launches ApplicationContainer monitors resources usage
    - cpu, memory, disk, network
  - Reports to RM

- **Per-application ApplicationMaster (AM)**
  - Negotiates resources from RM
  - Works with NMs to execute and monitor tasks
YARN Architecture

Job submission
Resource request
Node status
Job status
Highly Available Resource Manager

- Realized through a primary/backup architecture
  - Primary RM is Active
  - Backup RM is in Standby mode waiting to take over should anything happen to the Active

```
zookeeper / HDFS
```

```
Metadata (backup RM)  Metadata (primary RM)
```

```
Data service (nodemanager)
```

CS236620 Big Data Technology
Hadoop Map-Reduce

Source: http://hortonworks.com/hdp/
Historical Perspective

- **GFS 2003**
- **MapReduce 2004**
- **Yahoo Hadoop 2006**
- **Open Source 2008-9**
- **Hadoop 2.0 2013**
Recap: MR Paradigm

**Map** (doc)

(docid, tokens[]) = parse(doc)

**foreach** tok (tokens[])

**output** (tok, docid)

**Reduce** (tok, docids[])

**output** (tok, docids[])

Single record → Multiple <key, value> pairs

Aggregate values for the same key

In parallel across many data slices ...

In parallel across many keys ...

Task Parallelism

CS236620 Big Data Technology
MR job Example

every R starts computing after all M’s have completed

CS236620 Big Data Technology
Storage

- Job input and output are stored on HDFS
  - Replicated, reliable storage
  - A mapper consumes one or more input splits
  - A reducer typically creates one output split

- Intermediate files reside on local disks
  - Non-reliable
  - A mapper writes one file for each reducer
  - A reducer pulls one file from each mapper

Unless \#reducers=0
Mapper versus Reducer (1)

- Mapper (stages happen in parallel)
  - **Scan** (few large) input splits from HDFS [S]
  - **Compute** : window=1, typically selection/projection [U]
  - **Sort** : sort & write intermediate outputs [S]

- Reducer (stages happen sequentially)
  - **Shuffle** : copy (many small) intermediate files [S]
  - **Sort** : merge by key [S]
  - **Compute** : window = group size, typically aggregation [U]
    write-back the output to HDFS [S]

CS236620 Big Data Technology
Mapper versus Reducer (2)

- Mapper
  - Low fan-in
  - I/O bound (linear scan)
  - Might be CPU-bound

- Reducer
  - High fan-in
  - Network-bound (shuffle)
  - Might be RAM-bound (sort)

- Require different optimizations
- #mappers and #reducers are independent parameters
Map-Reduce Job Dynamics

- A job proceeds in execution waves
  - Typically, \#tasks >> \#machines

- Synchronous execution
  - Neither of the reducers can start the compute stage before all the mappers complete

- The shuffle and sort stages can overlap
  - Hadoop’s rule of thumb: start scheduling reducers when 75% of the mappers complete

- Data transfer rate over time
Anatomy of a Map-Reduce Job

Map Phase

Reduce Phase

INIT

EXECUTION

SHUFFLE (local)

SPILLING

Job Start

Job Finish

Task Start

Task Finish

time

set up map
clean

Source: http://ercoppa.github.io/HadoopInternals/AnatomyMapReduceJob.html

CS236620 Big Data Technology
Sorted output is split into partitions
(*) Execute combiner on each partition
Partitions are written to local file
4 reducers == 4 partitions

Source: [http://ercoppa.github.io/HadoopInternals/AnatomyMapReduceJob.html](http://ercoppa.github.io/HadoopInternals/AnatomyMapReduceJob.html)

CS236620 Big Data Technology
Mapper Optimization: Locality

- Let’s exploit HDFS’s redundancy in storage
  - Push the computation as close as possible to the replicas
  - → avoid data transfers over the network

- Applies only to mappers (why?)

- Implied design choice: non-tiered storage & compute

- Scheduler’s priorities (Hadoop)
  - Local replica, then rack-local replica, then all the rest
Combiners

- Often, the reducer does simple aggregation
  - Sum, average, min/max, ...
  - Commutative and associative functions

\[
\max(X \cup Y \cup Z) = \max\{\max(X), \max(Y), \max(Z)\}
\]

\[
\sum(X \cup Y \cup Z) = \sum X + \sum Y + \sum Z
\]

- We can do some aggregation at the mapper side
  - ... and eliminate a lot of network traffic!
- Extensively used by query languages on top of MR
Straggler Tasks

Slowest task (straggler) affects the job latency
Speculative Execution

- Schedule a **backup** task if the original task takes too long to complete
  - Same input(s), different output(s)
- Failed tasks and stragglers get the same treatment
- Let the fastest win
  - After one task completes, kill all the clones
- Challenge: how can we tell a task is late?
Illustration: Speculative Execution

Source: Google Map-Reduce paper, OSDI 2004

CS236620 Big Data Technology
Multi-Tenancy

- MR was not designed for multi-user systems
  - Hard to accommodate heterogeneous requirements
    - E.g., production batches vs research ad-hoc queries
    - No real resource protection
  - Take 1: build a cluster per user
  - Take 2: one platform to rule them all
    - Scales much better
    - Requires OS-style resource management
    - Either build support into MR, or use cloud virtualization
Summary

- Scalable computing stack by decoupling
  - Distributed filesystem
  - Job scheduling and resource management
- Master nodes (NN and RM) are highly available
- Multitenancy: supports multiple users and applications
- M and R have different nature
  - Optimized separately
- Not a silver bullet
  - Too primitive
  - Good only for batch processing – not for iterative modeling and graph processing
Further Reading

- Google Filesystem (GFS) paper
- Apache Hadoop
- Hadoop YARN
Next...

- Generalizing the MR paradigm
- Spark: in-memory distributed data processing