CC755: Distributed and Parallel Systems

Lecture 12: Hadoop Map Reduce

Slides by: Instructor: Nalini Venkatasubramanian
Class: CS 237 Distributed Systems Middleware

Dr. Manal Helal, Spring 2016

moodle.manalhelal.com

Introduction

1. **Introduction: Hadoop's history and advantages**

2. **Architecture in detail**

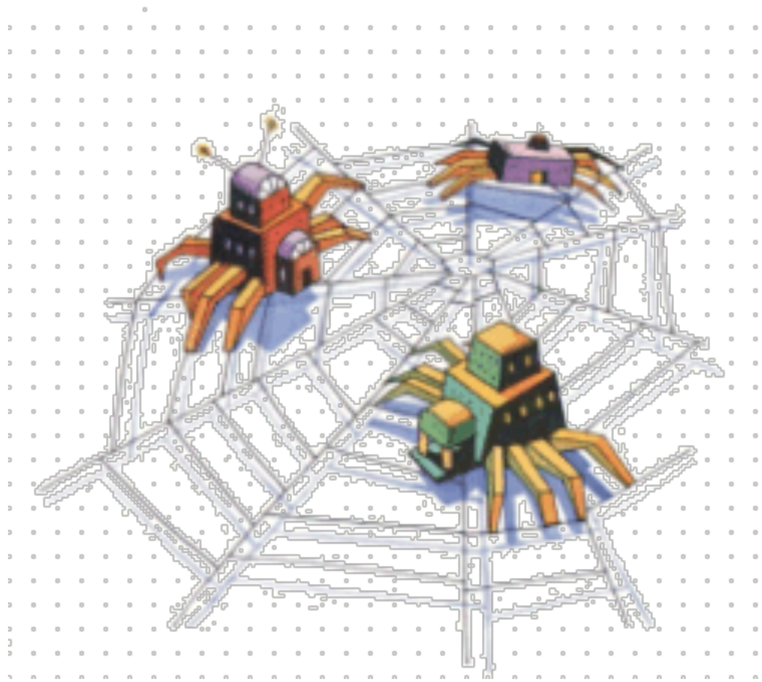
3. **Hadoop in industry**

What is Hadoop?

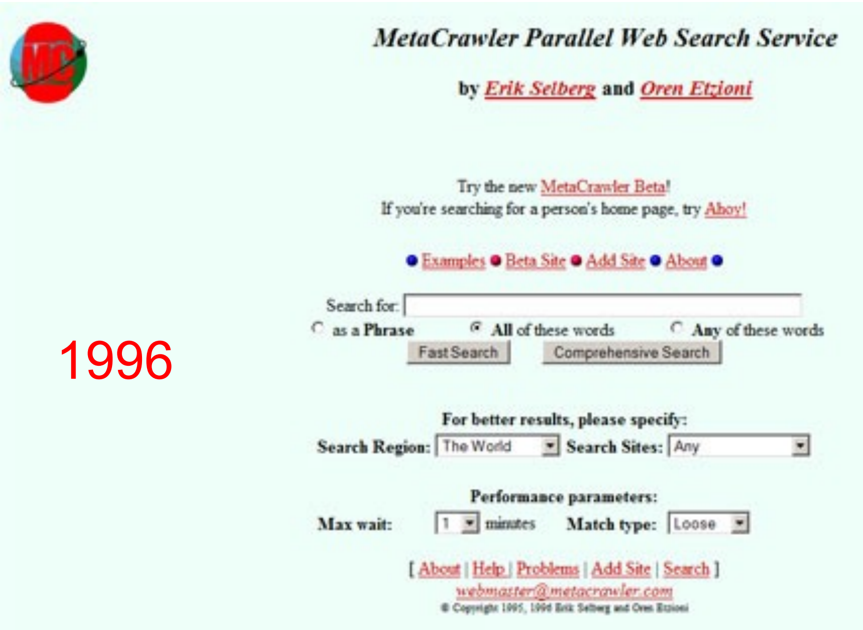
- Apache top level project, open-source implementation of frameworks for reliable, scalable, distributed computing and data storage.
- It is a flexible and highly-available architecture for large scale computation and data processing on a network of commodity hardware.

Brief History of Hadoop

- Designed to answer the question: **“How to process big data with reasonable cost and time?”**



Search engines in 1990s



1996

1996

1996

1997

Google search engines

1998

2013

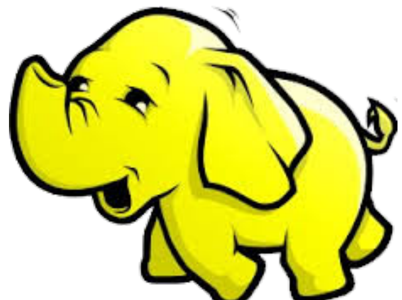
Hadoop's Developers

Doug Cutting

2005: Doug Cutting and Michael J. Cafarella developed Hadoop to support distribution for the Nutch search engine project.

The project was funded by Yahoo.

2006: Yahoo gave the project to Apache Software Foundation.



Google Origins

2003

The Google File System

Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung

Google*

→ hadoop HDFS

2004

MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

Google, Inc.

→ hadoop Map Reduce

2006

Bigtable: A Distributed Storage System for Structured Data

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach
Mike Burrows, Tushar Chandra, Andrew Fikes, Robert E. Gruber

{fay,jeff,sanjay,wilsonh,kerr,m3b,tushar,fikes,gruber}@google.com

Google, Inc.

Abstract

gtable is a distributed storage system for managing tured data that is designed to scale to a very large petabytes of data across thousands of commodity rs. Many projects at Google store data in Bigtable, ding web indexing, Google Earth, and Google Fi- e. These applications place very different demands igtable, both in terms of data size (from URLs to

achieved scalability and high performance, but Big provides a different interface than such systems. Big does not support a full relational data model; inste provides clients with a simple data model that sup dynamic control over data layout and format, an lows clients to reason about the locality properties o data represented in the underlying storage. Data i dexed using row and column names that can be arbi strings. Bigtable also treats data as uninterpreted str

→ APACHE HBASE

Some Hadoop Milestones

- **2008 - Hadoop Wins Terabyte Sort Benchmark (**sorted 1 terabyte of data in 209 seconds, compared to previous record of 297 seconds)
- 2009 - Avro and Chukwa became new members of Hadoop Framework family
- 2010 - Hadoop's Hbase, Hive and Pig subprojects completed, adding more computational power to Hadoop framework
- **2011 - ZooKeeper Completed**
- **2013 - Hadoop 1.1.2 and Hadoop 2.0.3 alpha.**
  - Ambari, Cassandra, Mahout have been added

What is Hadoop?

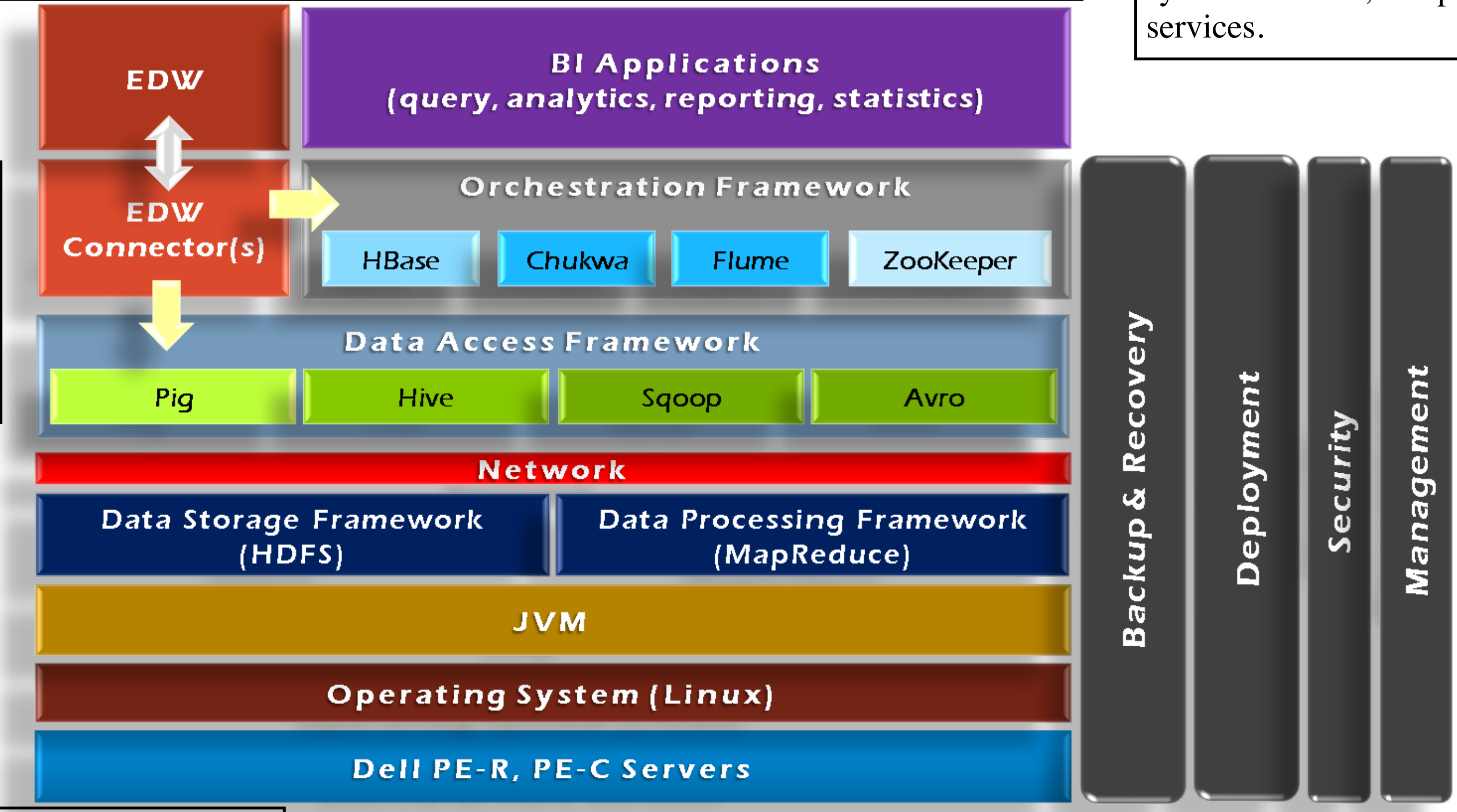
- **Hadoop:**
  - an open-source software framework that supports data-intensive distributed applications, licensed under the Apache v2 license.
- **Goals / Requirements:**
  - Abstract and facilitate the storage and processing of large and/or rapidly growing data sets
    - Structured and non-structured data
    - Simple programming models
  - High scalability and availability
  - Use commodity (cheap!) hardware with little redundancy
  - Fault-tolerance
  - Move computation rather than data

Hadoop Framework Tools

Flume is a distributed, reliable, and available service for efficiently collecting, aggregating, and moving large amounts of log data.

ZooKeeper is a centralized service for maintaining configuration information, naming, providing distributed synchronization, and providing group services.

HBase is the Hadoop database, a distributed, scalable, big data store.



Chukwa is an open source data collection system for monitoring large distributed systems

Avro™ is a data serialization system.

Sqoop is a tool designed for efficiently transferring bulk data between Apache Hadoop and structured datastores such as relational databases.

Hive is a data warehouse software facilitates reading, writing, and managing large datasets residing in distributed storage using SQL.

Pig is a platform for analyzing large data sets that consists of a high-level language for expressing data analysis programs,

Hadoop Installation & Tutorials

Download From: http://hadoop.apache.org/releases.html
Single Node Setup: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/SingleCluster.html
Cluster Setup: http://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-common/ClusterSetup.html
Map-Reduce Tutorial: http://hadoop.apache.org/docs/current/hadoop-mapreduce-client/hadoop-mapreduce-client-core/MapReduceTutorial.html

Common Hadoop Shell Commands

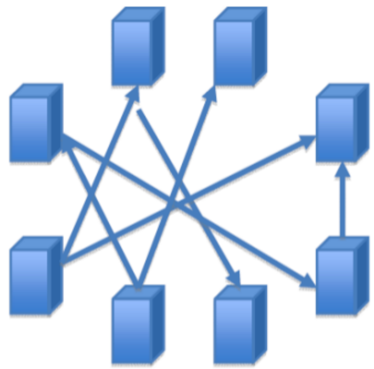
```
hadoop fs –cat file:///file2
hadoop fs –mkdir /user/hadoop/dir1 /user/hadoop/dir2
hadoop fs –copyFromLocal <fromDir> <toDir>
hadoop fs –put <localfile> hdfs://nn.example.com/hadoop/
   hadoopfile
sudo hadoop jar <jarFileName> <method> <fromDir>
   <toDir>
hadoop fs –ls /user/hadoop/dir1

hadoop fs –cat hdfs://nn1.example.com/file1
hadoop fs –get /user/hadoop/file <localfile>
```

Tips
-- 'sudo' means 'run as administrator' (super user)
--some hadoop configurations use 'hadoop dfs' rather than 'hadoop fs' – file paths to hadoop differ for the former, see the link included for more detail

Understanding MapReduce

- **Map>>**
  - (K1, V1) →
    - Info in
    - Input Split
  - list (K2, V2)
    - Key / Value out (intermediate values)
    - One list per local node
    - Can implement local Reducer (or Combiner)

- **Shuffle/Sort>>**



- **Reduce**
  - (K2, list(V2) →
    - Shuffle / Sort phase precedes Reduce phase
    - Combines Map output into a list
  - list (K3, V3)
    - Usually aggregates intermediate values

(input) <k1, v1> → **map** → <k2, v2> → **combine** → <k2, v2> → **reduce** → <k3, v3> *(output)*

MapReduce Example - WordCount

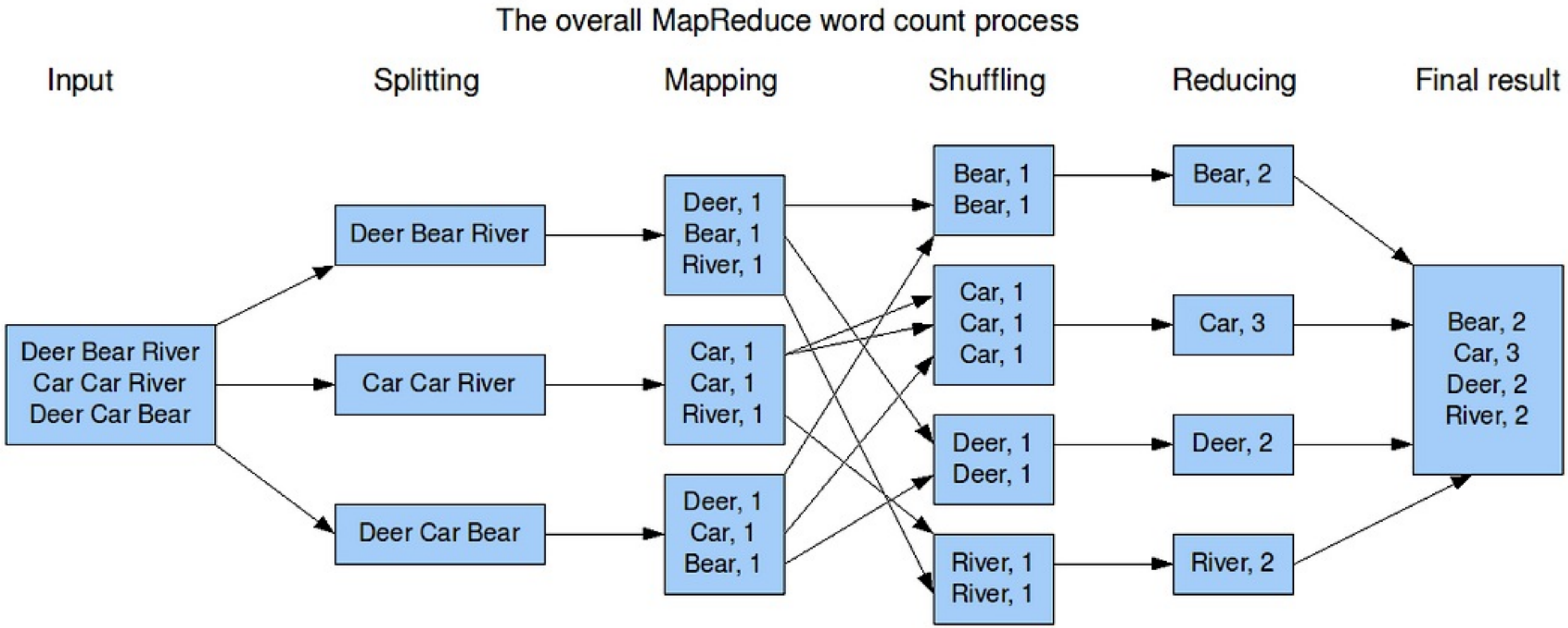


Image from: http://blog.jteam.nl/wp-content/uploads/2009/08/MapReduceWordCountOverview1.png

Hadoop’s Architecture

- Distributed, with some centralization
- Main nodes of cluster are where most of the computational power and storage of the system lies
- Main nodes run TaskTracker to accept and reply to MapReduce tasks, and also DataNode to store needed blocks closely as possible
- Central control node runs NameNode to keep track of HDFS directories & files, and JobTracker to dispatch compute tasks to TaskTracker
- Written in Java, also supports Python and Ruby

Ways to MapReduce

Libraries

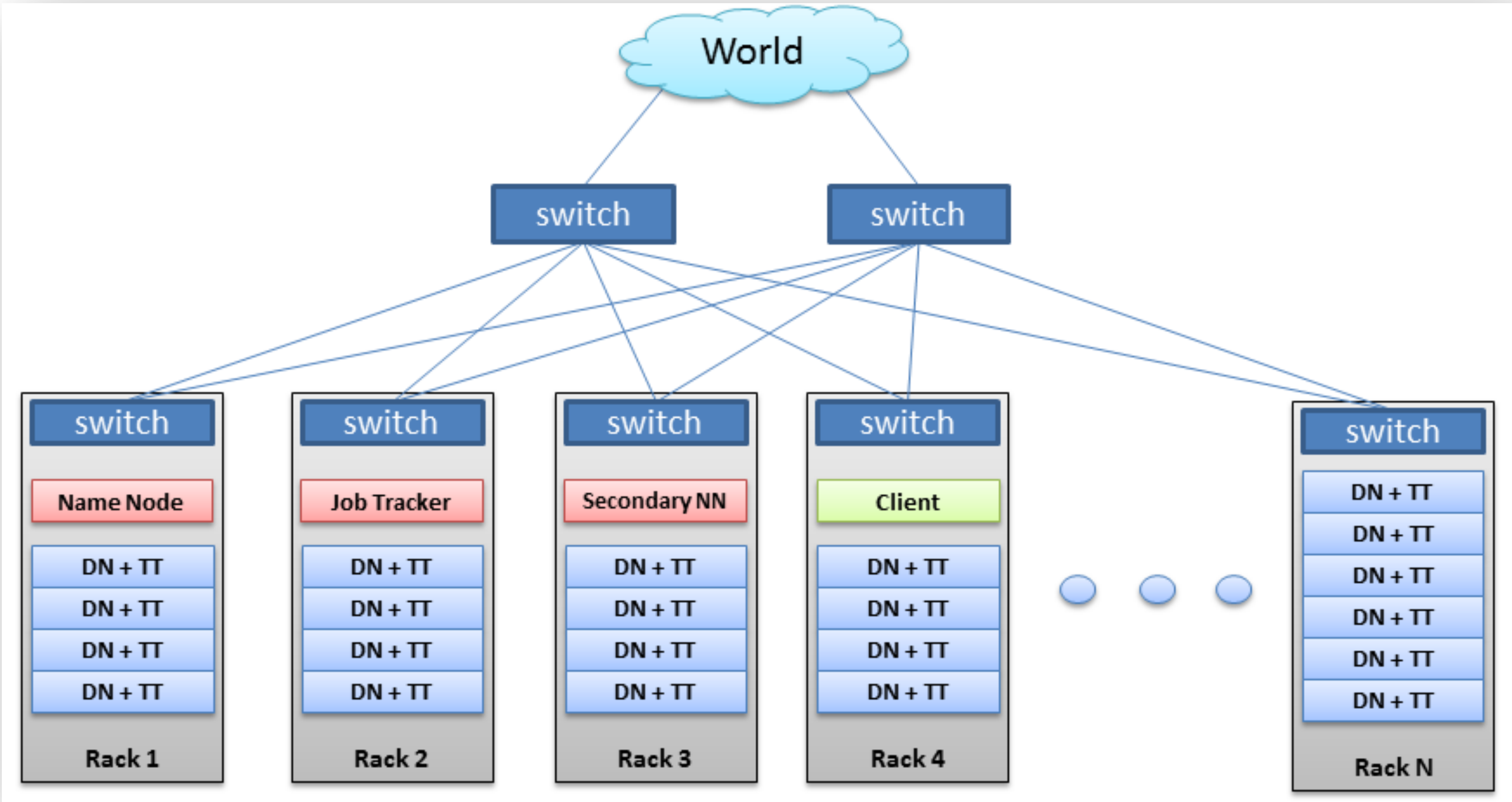
- HBase
- Hive
- Pig
- Sqoop
- Oozie
- Mahout
- Others…

Languages

- Java*
- HiveQL (HQL)
- Pig Latin
- Python
- C#
- JavaScript
- R
- More…

Note: Java is most common, but other languages can be used

Hadoop's Architecture



Hadoop's Architecture

- Hadoop Distributed Filesystem
- Tailored to needs of MapReduce
- Targeted towards many reads of filestreams
- Writes are more costly
- High degree of data replication (3x by default)
- No need for RAID on normal nodes
- Large blocksize (64MB)
- Location awareness of DataNodes in network

Hadoop’s Architecture

NameNode:

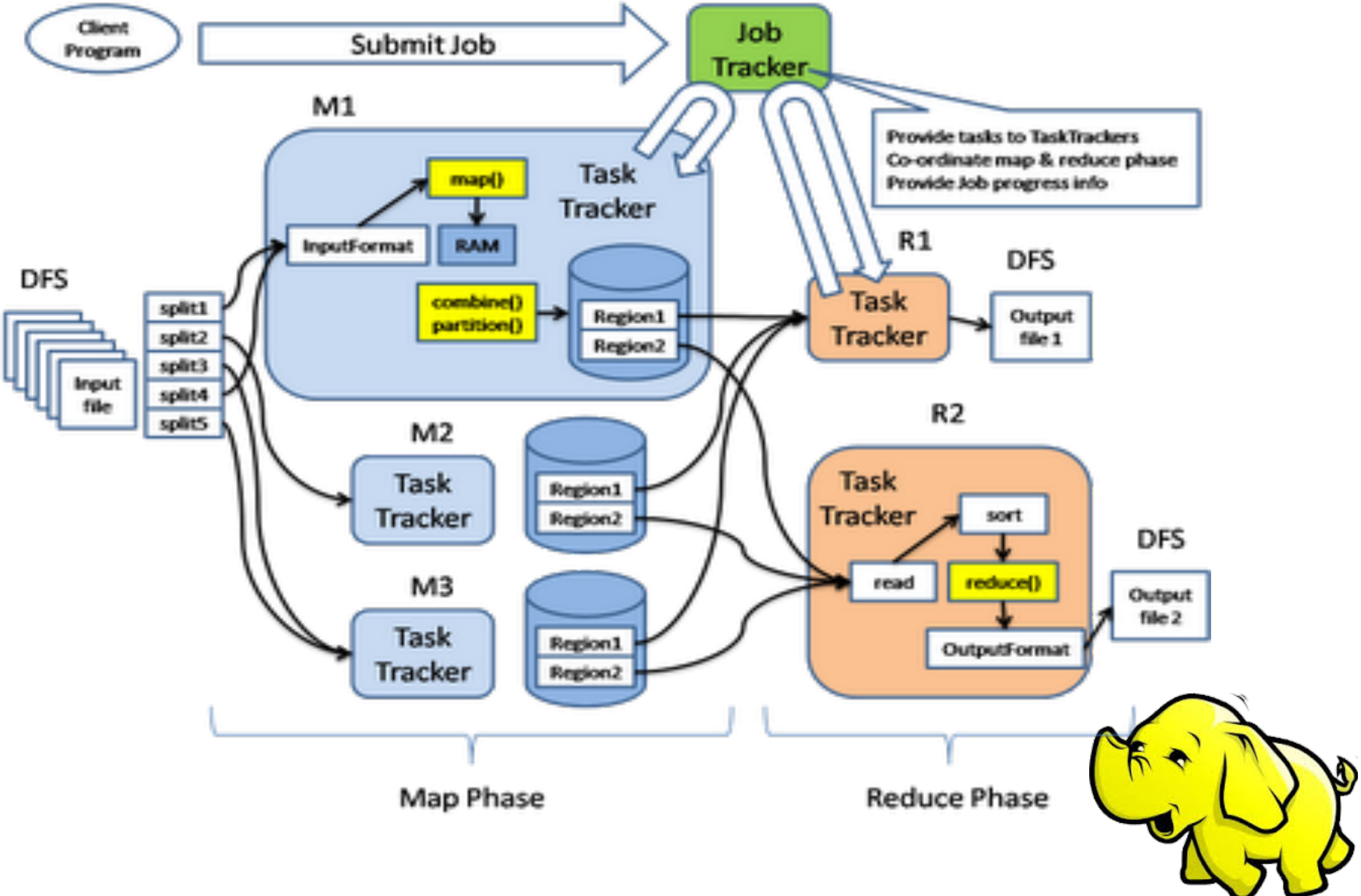
- Stores metadata for the files, like the directory structure of a typical FS.
- The server holding the NameNode instance is quite crucial, as there is only one.
- Transaction log for file deletes/adds, etc. Does not use transactions for whole blocks or file-streams, only metadata.
- Handles creation of more replica blocks when necessary after a DataNode failure

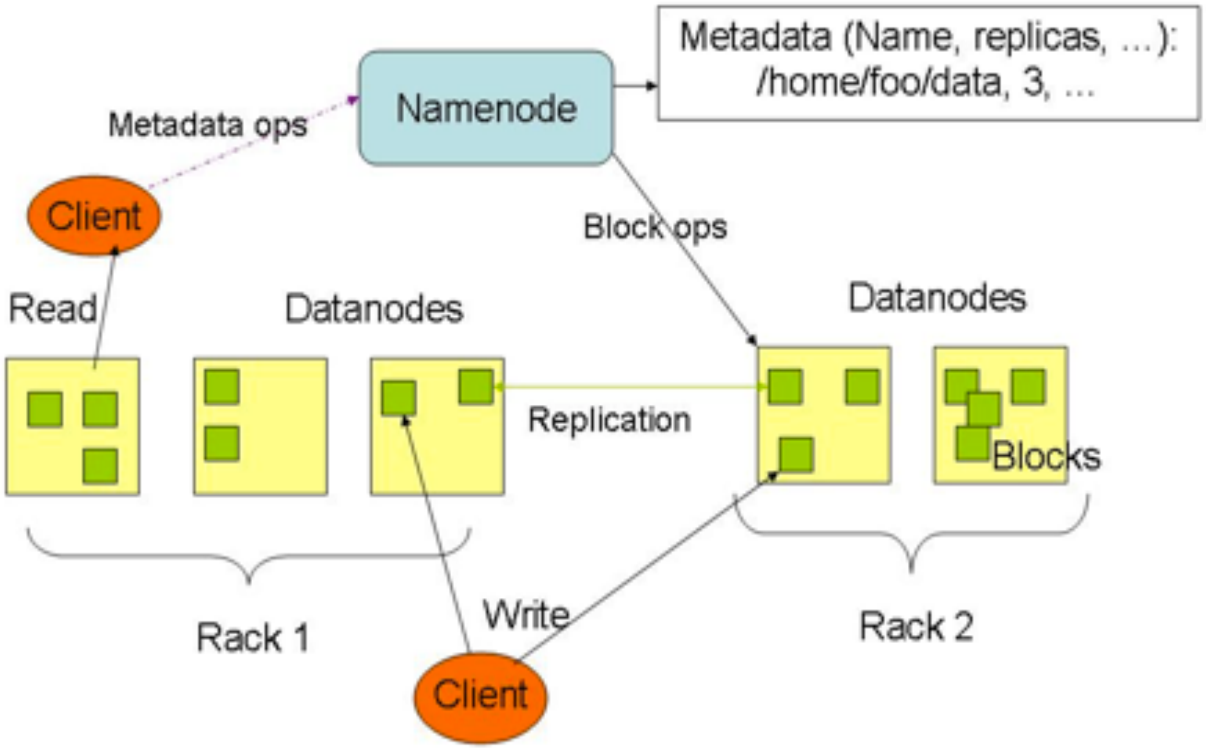
Hadoop's Architecture

DataNode:

- Stores the actual data in HDFS
- Can run on any underlying filesystem (ext3/4, NTFS, etc)
- Notifies NameNode of what blocks it has
- NameNode replicates blocks 2x in local rack, 1x elsewhere

Hadoop's Architecture: MapReduce Engine




Metadata (Name, replicas, ...):
/home/foo/data, 3, ...
Namenode
Metadata ops
Client
Block ops
Read
Datanodes
Datanodes
Replication
Blocks
Rack 1
Write
Rack 2
Client

Hadoop's Architecture

MapReduce Engine:

- JobTracker & TaskTracker
- JobTracker splits up data into smaller tasks("Map") and sends it to the TaskTracker process in each node
- TaskTracker reports back to the JobTracker node and reports on job progress, sends data ("Reduce") or requests new jobs

Hadoop’s Architecture

- None of these components are necessarily limited to using HDFS
- Many other distributed file-systems with quite different architectures work
- Many other software packages besides Hadoop's MapReduce platform make use of HDFS

Hadoop in the Wild

- Hadoop is in use at most organizations that handle big data:
  - Yahoo!
  - Facebook
  - Amazon
  - Netflix
  - Etc…

- Some examples of scale:
  - Yahoo!'s Search Webmap runs on 10,000 core Linux cluster and powers Yahoo! Web search

  - FB's Hadoop cluster hosts 100+ PB of data (July, 2012) & growing at ½ PB/day (Nov, 2012)

Hadoop in the Wild

Three main applications of Hadoop:

- Advertisement (Mining user behavior to generate recommendations)
- Searches (group related documents)
- Security (search for uncommon patterns)

Hadoop in the Wild

- Non-realtime large dataset computing:
    - NY Times was dynamically generating PDFs of articles from 1851-1922
    - Wanted to pre-generate & statically serve articles to improve performance
    - Using Hadoop + MapReduce running on EC2 / S3, converted 4TB of TIFFs into 11 million PDF articles in 24 hrs

Hadoop in the Wild: Facebook Messages

- Design requirements:
  - Integrate display of email, SMS and chat messages between pairs and groups of users
  - Strong control over who users receive messages from
  - Suited for production use between 500 million people immediately after launch
  - Stringent latency & uptime requirements

Hadoop in the Wild

- System requirements
  - High write throughput
  - Cheap, elastic storage
  - Low latency
  - High consistency (within a single data center good enough)
  - Disk-efficient sequential and random read performance

Hadoop in the Wild

- Classic alternatives
  - These requirements typically met using large MySQL cluster & caching tiers using Memcached
  - Content on HDFS could be loaded into MySQL or Memcached if needed by web tier
- Problems with previous solutions
  - MySQL has low random write throughput… BIG problem for messaging!
  - Difficult to scale MySQL clusters rapidly while maintaining performance
  - MySQL clusters have high management overhead, require more expensive hardware

Hadoop in the Wild

- Facebook’s solution
  - Hadoop + HBase as foundations
  - Improve & adapt HDFS and HBase to scale to FB’s workload and operational considerations
    - Major concern was availability: NameNode is SPOF & failover times are at least 20 minutes
    - Proprietary “AvatarNode”: eliminates SPOF, makes HDFS safe to deploy even with 24/7 uptime requirement
    - Performance improvements for realtime workload: RPC timeout. Rather fail fast and try a different DataNode

Hadoop Highlights

- Distributed File System
- Fault Tolerance
- Open Data Format
- Flexible Schema
- Queryable Database

Why use Hadoop?

- Need to process Multi Petabyte Datasets
- Data may not have strict schema
- Expensive to build reliability in each application
- Nodes fails everyday
- Need common infrastructure
- Very Large Distributed File System
- Assumes Commodity Hardware
- Optimized for Batch Processing
- Runs on heterogeneous OS

What types of business problems for Hadoop?

Risk Modeling	Customer Churn Analysis	Recommendation Engine
Ad Targeting	Point of Sale Transactional Analysis	Threat Analysis
Trade Surveillance	Search Quality	Data Sandbox

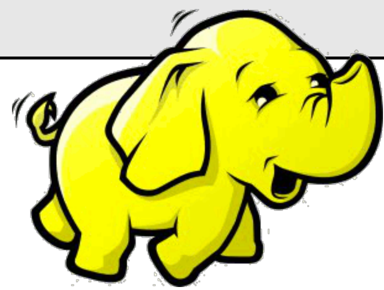
Source: Cloudera “Ten Common Hadoopable Problems”

Limitations of MapReduce

- Batch processing, not interactive
- Designed for a specific problem domain
- MapReduce programming paradigm not commonly understood (functional)
- Lack of trained support professionals
- API / security model are moving targets

Comparing: RDBMS vs. Hadoop

	Traditional RDBMS	Hadoop / MapReduce
Data Size	Gigabytes (Terabytes)	Petabytes (Hexabytes)
Access	Interactive and Batch	Batch – NOT Interactive
Updates	Read / Write many times	Write once, Read many times
Structure	Static Schema	Dynamic Schema
Integrity	High (ACID)	Low
Scaling	Nonlinear	Linear
Query Response Time	Can be near immediate	Has latency (due to batch processing)



More about the operations of each process

DataNode

- A Block Sever
  - Stores data in local file system
  - Stores meta-data of a block - checksum
  - Serves data and meta-data to clients
- Block Report
  - Periodically sends a report of all existing blocks to NameNode
- Facilitate Pipelining of Data
  - Forwards data to other specified DataNodes

Block Placement

- Replication Strategy
  - One replica on local node
  - Second replica on a remote rack
  - Third replica on same remote rack
  - Additional replicas are randomly placed
- Clients read from nearest replica

Data Correctness

- Use Checksums to validate data – CRC32
- File Creation
  - Client computes checksum per 512 byte
  - DataNode stores the checksum
- File Access
  - Client retrieves the data and checksum from DataNode
  - If validation fails, client tries other replicas

Data Pipelining

- Client retrieves a list of DataNodes on which to place replicas of a block
- Client writes block to the first DataNode
- The first DataNode forwards the data to the next DataNode in the Pipeline
- When all replicas are written, the client moves on to write the next block in file

Hadoop MapReduce

- MapReduce programming model
  - Framework for distributed processing of large data sets
  - Pluggable user code runs in generic framework
- Common design pattern in data processing
  - cat * | grep | sort | uniq -c | cat > file
  - input | **map** | shuffle | **reduce** | output

MapReduce Usage

- Log processing
- Web search indexing
- Ad-hoc queries

Closer Look

- MapReduce Component
  - JobClient
  - JobTracker
  - TaskTracker
  - Child
- Job Creation/Execution Process

MapReduce Process (org.apache.hadoop.mapred)

- JobClient
  - Submit job
- JobTracker
  - Manage and schedule job, split job into tasks
- TaskTracker
  - Start and monitor the task execution
- Child
  - The process that really execute the task

Inter Process Communication
IPC/RPC (org.apache.hadoop.ipc)

- Protocol
  JobSubmissionProtocol
  - JobClient <-------------> JobTracker

  InterTrackerProtocol
  - TaskTracker <------------> JobTracker

  TaskUmbilicalProtocol
  - TaskTracker <-------------> Child
- JobTracker implements both protocol and works as server in both IPC
- TaskTracker implements the TaskUmbilicalProtocol; Child gets task information and reports task status through it.

The Following Slides are for interfering with the Hadoop echo system to directly manage the splitting process and the tasks management

For end user Examples, check the other slides and the accompanying source code on moodle

JobClient.submitJob - 1

- Check input and output, e.g. check if the output directory is already existing
  - job.getInputFormat().validateInput(job);
  - job.getOutputFormat().checkOutputSpecs(fs, job);
- Get InputSplits, sort, and write output to HDFS
  - InputSplit[] splits = job.getInputFormat().
    getSplits(job, job.getNumMapTasks());
  - writeSplitsFile(splits, out); // out is $SYSTEMDIR/$JOBID/job.split

Hadoop divides the input file stored on HDFS into splits (typically of the size of an HDFS block) and assigns every split to a different mapper, trying to assign every split to the mapper where the split physically resides

JobClient.submitJob - 2

- The jar file and configuration file will be uploaded to HDFS system directory
  - job.write(out); // out is $SYSTEMDIR/$JOBID/job.xml
- JobStatus status = jobSubmitClient.submitJob(jobId);
  - This is an RPC invocation, jobSubmitClient is a proxy created in the initialization

Job initialization on JobTracker - 1

- JobTracker.submitJob(jobID) <-- receive RPC invocation request
- JobInProgress job = new JobInProgress(jobId, this, this.conf)
- Add the job into Job Queue
  - jobs.put(job.getProfile().getJobId(), job);
  - jobsByPriority.add(job);
  - jobInitQueue.add(job);

Job initialization on JobTracker - 2

- Sort by priority
  - resortPriority();
  - compare the JobPrioity first, then compare the JobSubmissionTime
- Wake JobInitThread
  - jobInitQueue.notifyall();
  - job = jobInitQueue.remove(0);
  - job.initTasks();

JobInProgress - 1

- JobInProgress(String jobid, JobTracker jobtracker, JobConf default_conf);
- JobInProgress.initTasks()
  - DataInputStream splitFile = fs.open(new Path(conf.get(“mapred.job.split.file”)));

    // mapred.job.split.file --> $SYSTEMDIR/
$JOBID/job.split

JobInProgress - 2

- splits = JobClient.readSplitFile(splitFile);
- numMapTasks = splits.length;
- maps[i] = new TaskInProgress(jobId, jobFile, splits[i], jobtracker, conf, this, i);
- reduces[i] = new TaskInProgress(jobId, jobFile, splits[i], jobtracker, conf, this, i);
- JobStatus --> JobStatus.RUNNING

JobTracker Task Scheduling - 1

- Task getNewTaskForTaskTracker(String taskTracker)
- Compute the maximum tasks that can be running on taskTracker
  - int maxCurrentMap Tasks = tts.getMaxMapTasks();
  - int maxMapLoad = Math.min(maxCurrentMapTasks, (int)Math.ceil(double) remainingMapLoad/ numTaskTrackers));

JobTracker Task Scheduling - 2

- int numMaps = tts.countMapTasks(); // running tasks number
- If numMaps < maxMapLoad, then more tasks can be allocated, then based on priority, pick the first job from the jobsByPriority Queue, create a task, and return to TaskTracker
  - Task t = job.obtainNewMapTask(tts, numTaskTrackers);

Start TaskTracker - 1

- initialize()
  - Remove original local directory
  - RPC initialization
    - TaskReportServer = RPC.getServer(this, bindAddress, tmpPort, max, false, this, fConf);
    - InterTrackerProtocol jobClient = (InterTrackerProtocol) RPC.waitForProxy(InterTrackerProtocol.class, InterTrackerProtocol.versionID, jobTrackAddr, this.fConf);

Start TaskTracker - 2

- run();
- offerService();
- TaskTracker talks to JobTracker with HeartBeat message periodically
  - HeatbeatResponse heartbeatResponse = transmitHeartBeat();

- TaskTracker.localizeJob(TaskInProgress tip);
- launchTasksForJob(tip, new JobConf(rjob.jobFile));
  - tip.launchTask(); // TaskTracker.TaskInProgress
  - tip.localizeTask(task); // create folder, symbol link
  - runner = task.createRunner(TaskTracker.this);
  - runner.start(); // start TaskRunner thread

- TaskRunner.run();
  - Configure child process' jvm parameters, i.e. classpath, taskid, taskReportServer's address & port
  - Start Child Process
    - runChild(wrappedCommand, workDir, taskid);

Child.main()

- Create RPC Proxy, and execute RPC invocation
  - TaskUmbilicalProtocol umbilical = (TaskUmbilicalProtocol) RPC.getProxy(TaskUmbilicalProtocol.class, TaskUmbilicalProtocol.versionID, address, defaultConf);
  - Task task = umbilical.getTask(taskid);
- task.run(); // mapTask / reduceTask.run

Finish Job - 1

- Child
  - task.done(umbilical);
    - RPC call: umbilical.done(taskId, shouldBePromoted)
- TaskTracker
  - done(taskId, shouldPromote)
    - TaskInProgress tip = tasks.get(taskid);
    - tip.reportDone(shouldPromote);
      - taskStatus.setRunState(TaskStatus.State.SUCCEEDED)

Finish Job - 2

- JobTracker
  - TaskStatus report: status.getTaskReports();
  - TaskInProgress tip = taskidToTIPMap.get(taskId);
  - JobInProgress update JobStatus
    - tip.getJob().updateTaskStatus(tip, report, myMetrics);
      - One task of current job is finished
      - completedTask(tip, taskStatus, metrics);
      - If (this.status.getRunState() == JobStatus.RUNNING && allDone) {this.status.setRunState(JobStatus.SUCCEEDED)}

Demo

- Word Count
  - hadoop jar hadoop-0.20.2-examples.jar wordcount <input dir> <output dir>
- Hive
  - hive -f pagerank.hive