The need for Big Data

Let’s look at some examples.

- Facebook currently stores 300 petabytes of data
- Facebook processes approximately 600 terabytes of data per day
- Facebook handles one billion users per day
- Facebook processes 2.7 billion ‘Likes’ per day.
- 300 million photographs uploaded every day.

The need for Big Data

- The United States National Security Agency (NSA)
- The NSA currently stores approximately five exabytes of data
- The NSA processes approximately 30 petabytes of data per day
- The NSA touches approximately 1.6 percent of all internet traffic per day (More than Google).
- Data such as web searches, phone calls, credit card transactions, health information is stored and analyzed.

The need for Big Data

- Google.
- Google processes 100 petabytes of information per day
- Google indexes 60 trillion web pages per day
- Google has greater than one billion unique searches per month
- Google processes 2.3 million searches per second.

The need for Big Data

No single hardware platform can handle data sets this large.

Big Data Requirements

To process this volume of data you need three things.

- The ability to store massive quantities of data
- The ability to extract and process elements of this data in a timely fashion.
- The ability for the system to scale as the volumes of data increase.

Examples of scalability requirements

LinkedIn grew from 37 million users in Q1 2009 to 450 million users in Q2 2016.

No single hardware platform can match these scalability requirements, no matter how much memory, CPU and storage is allocated to it.

History of Big Data systems.

Google developed this concept in the early 2000's after realizing that the web was growing too quickly for existing architectures to process it.

Google created two systems to address this problem.

- The Google File System to solve the problem of distributed storage
- The MapReduce programming architecture. to solve the problem of distributed processing.

History of Big Data systems.

The Apache foundation took Google's technologies and developed an open source version.
The Google File System became HDFS
The MapReduce architecture became Hadoop's Mapreduce
Hadoop is the name for the overall system architecture.
It includes HDFS, MapReduce and YARN.

- b

History of Big Data systems.

Hadoop 2.0 released by the Apache Foundation in 2013.
Hadoop 2.0 is a fundamental change in the architecture.
The resource manager YARN was split from the MapReduce system and created as its own subsystem.

- b

History of Big Data systems.

Hadoop 2.0 released by the Apache Foundation in 2013.
Hadoop 2.0 is a fundamental change in the architecture.
The resource manager YARN was split from the MapReduce system and created as its own subsystem.

Hadoop Ecosystem

Hive

Hbase

Pig

Hadoop

Flume/Sqoop

Spark

Oozle

Hive

Provides an SQL interface to Hadoop

The bridge to Hadoop for people who don’t have experience with OOP in Java

Hbase

A database management system on top of Hadoop

Integrates with your application as if it were a traditional database

Pig

A data manipulation language

Transforms unstructured data into a structured format

Query the structured data using interfaces like Hive.

Spark

A distributed computing engine used along with Hadoop

Interactive shell used to quickly process data

Many built-in libraries for machine learning, NLP, stream processing, and other tasks.

Oozle

Workflow scheduling engine

Flume/Sqoop

Data transfer engine between Hadoop and other types of systems.

HDFS Design

- Each HDFS data node is a simple off-the-shelf system.
- HDFS is highly fault tolerant . Hardware failures are assumed.
- HDFS is designed for batch processing, HDFS is used for high throughput requirements rather than low latency requirements.
- Queries to HDFS are not usually done in real time.
- HDFS supports very large data set sizes.

HDFS Design

- Default block size for Hadoop is 64 Mb.
- Default blocksize for the Cloudera distribution is 128 Mb
- Files are split among multiple machines and disk storage in a cluster.
- One of the machines in a cluster is the master (name) node,
- The rest of them are data nodes.

HDFS Design

•The name node stores the meta data for all files stored in the HDFS.
•There is one name node per cluster and possibly a secondary name node.
•The name node stores all of the metadata for files.
•Consider a file lik e a book. The name node stores the index
•The data nodes store the actual pages.

HDFS Design

•The name node stores the meta data for all files stored in the HDFS.
•There is one name node per cluster and possibly a secondary name node.
•The name node stores all of the metadata for files.
•Consider a file lik e a book. The name node stores the index
•The data nodes store the actual pages.

HDFS Design

- The name node has two primary responsibilities.
- First, it manages the HD file system. All requests from a client for data go to the name node first.
- The name node stores the file system directory structure and all other metadata about each file.
- The data node actually stores the data in blocks on its disk storage devices.

- b

HDFS Design

- A typical data file is a large text file (size in terabytes or petabytes).
- This file is broken up into chunks called blocks.
- Each block is stored on a different node in the hadoop cluster.
- Each block is the same size.
- Differently sized files are treated the same way.
- Storage is simplified. We only deal with blocks.
- A single unit for fault tolerance and replication.

HDFS Design

- The default block size in Hadoop is 64 MB (128 MB for Cloudera installations).
- Block size is a trade off between parallelism and overhead.
- Large block sizes means less processes working. Each process only works on one block
- Smaller block sizes means more network and processing overhead.
- Optimal blocksize depends on type of data and other

HDFS Design

- The default block size in Hadoop is 64 MB (128 MB for Cloudera installations).
- Block size is a trade off between parallelism and overhead.
- Large block sizes means less processes working. Each process only works on one block
- Smaller block sizes means more network and processing overhead.
- Optimal blocksize depends on type of data and other considerations.
- Hadoop does its best to run all its tasks on the system that stores

HDFS Design

- The default block size in Hadoop is 64 MB (128 MB for Cloudera installations).
- Block size is a trade off between parallelism and overhead.
- Large block sizes means less processes working. Each process only works on one block
- Smaller block sizes means more network and processing overhead.
- Optimal blocksize depends on type of data and other considerations.
- Hadoop does its best to run all its tasks on the system that stores the block. This is data locality optimization

HDFS Design

- Hadoop nodes are generally implemented as commodity hardware.
- Commodity hardware can fail.
- We have to consider possibility of failure in the name node and data nodes.
- Some data in a block may get corrupted.
- A data node could crash completely, causing us to lose access to all the data stored.

HDFS Design

- Hadoop replicates every file and block stored in the HDFS.
- We define a replication factor to decide how data is replicated.
- Replication locations are also stored in the name node.
- Replication strategy needs to optimize two variables
- First, the redundancy factor. A higher redundancy factor means more security.
- Second, the write bandwidth. A higher redundancy factor also means a higher write time.

HDFS Design

- Nodes on different racks are further away than nodes on the same rack.
- A cluster of machines is made up of systems stored on different racks in a system room.
- Read/write bandwidth for inter rack nodes are lower than for intra rack nodes.
- The default replication strategy is to replicate data on one node as close to the same rack as possible.
- Hadoop then chooses another rack in a different location and writes the second replica to it.
- The third replica is on the new rack (but on a different node) as well,
- Replication strategy is defined in hdfs-site.xml configuration file.

HDFS Design

- A cluster of machines is made up of systems stored on different racks in a system room.
- Nodes on different racks are further away than nodes on the same rack.
- Read/write bandwidth for inter rack nodes are lower than for intra rack nodes.
- The default replication strategy is to replicate data on one node as close to the same rack as possible.
- Hadoop then chooses another rack in a different location and writes the second replica to it.
- The third replica is on the new rack (but on a different node) as well,
- Replication strategy is defined in hdfs-site.xml configuration file.

HDFS Design

- Fault tolerance planning must also take into consideration failure of the name node.
- Name node failure means that all of the data in the cluster is permanently lost.
- With name node failure, all file/block mapping is gone.
- Two specific strategies for overcoming name node failure.
- First is the metadata files.
- Second is the secondary name node.

HDFS Design

- Two metadata files associated with the name node.
- First is the *fsimage* File system image.
- Second is the edit log.

HDFS Design

- The fsimage file holds a snapshot of the hadoop cluster's file system on startup.
- It contains the image of the file system before any writes have been performed.
- FSImage is loaded into memory on startup.

HDFS Design

- The edit log contains a log of all modifications to files in the cluster's file system since startup.
- Putting together the fsimage and the edit log can reconstruct the data in the hdfs cluster.
- Both files are by default saved to a local file system directory on the name node.
- Alternatively (or simultaneously) this data can be saved to a remote location over the network.
- Merging these two files can be computationally intensive.

HDFS Design

- The secondary name node contains a replica of the fsimage and edit log from the master.
- The secondary namenode merges the files together according to a specifically defined checkpoint.
- The newly merged fsimage is then copied back to the primary name node.
- The edits log on both nodes is than reset.

HDFS Design

- Two ways to define checkpointing.
- First is to checkpoint after a specific amount of time.
- Second is to checkpoint after a specific number of transactions are completed.
- The newly merged fsimage is then copied back to the primary name node.
- The edits log on both nodes is than reset.

HDFS Installation

Hadoop Installation Modes

Standalone

Pseudo-distributed

Fully distributed

Standalone

The default mode for Hadoop
Runs on a single node
A single JVM process
YARN and HDFS do not run
Used to test Mapreduce programs

Pseudo-distributed

Runs on a single node

Uses two JVM processes to simulate a distributed environment

HDFS for storage

YARN for managing tasks

Used as a fully-fledged test environment

Fully distributed

Runs on a cluster of machines, generally Linux systems in a data center or Virtual Machines in a cloud environment

Usually implemented via Cloudera, Hortonworks or MapR rather than manually.

MapReduce

- Mapreduce jobs require running processes on many different machines.
- Mapreduce is a programming paradigm that allows processing on distributed systems.
- Modern systems generate millions of records of raw data.
- This data can be many petabytes in size.
- These sort of tasks are run in two phases, the *map* phase and the *reduce* phase.

MapReduce

- The map task divides a job into many different processes, each running on a different compute node.
- The reduce task takes the output of the many map tasks and runs some sort of reducing function on it to generate the desired output.
- The developer only needs to write a map and reduce function. All of the fault tolerance is handled by hadoop itself.
- Map output is in the form of *key/value* pairs.
- The reduce function takes these key/value pairs and runs a reducing function to get the desired output to the client,.

MapReduce

•In order to run a mapreduce job on the Hadoop cluster, the job must be compiled into a JAR file.

•This jar file is then copied to the designated Hadoop distribution directory for processing.

•A Mapreduce input directory needs to be created in the cluster's HDFS.

•To actually run a hadoop job, use the *hadoop jar* command.

Mapreduce

- When a job has been submitted, you will see a job id starting with 'job_id<xxxxx>'
- Using this job ID, you can monitor the results of the hadoop job .
- Hadoop provides a web based resource manager interface .
- The default URL is *http://localhost:8088*.

YARN

- YARN stands for Yet Another Resource Negotiator.
- In Hadoop Version 1, this was part of the MapReduce framework.
- Hadoop Version 2 split the resource management portion into its own subsystem.
- Yarn is used by Hadoop to schedule tasks for the cluster.

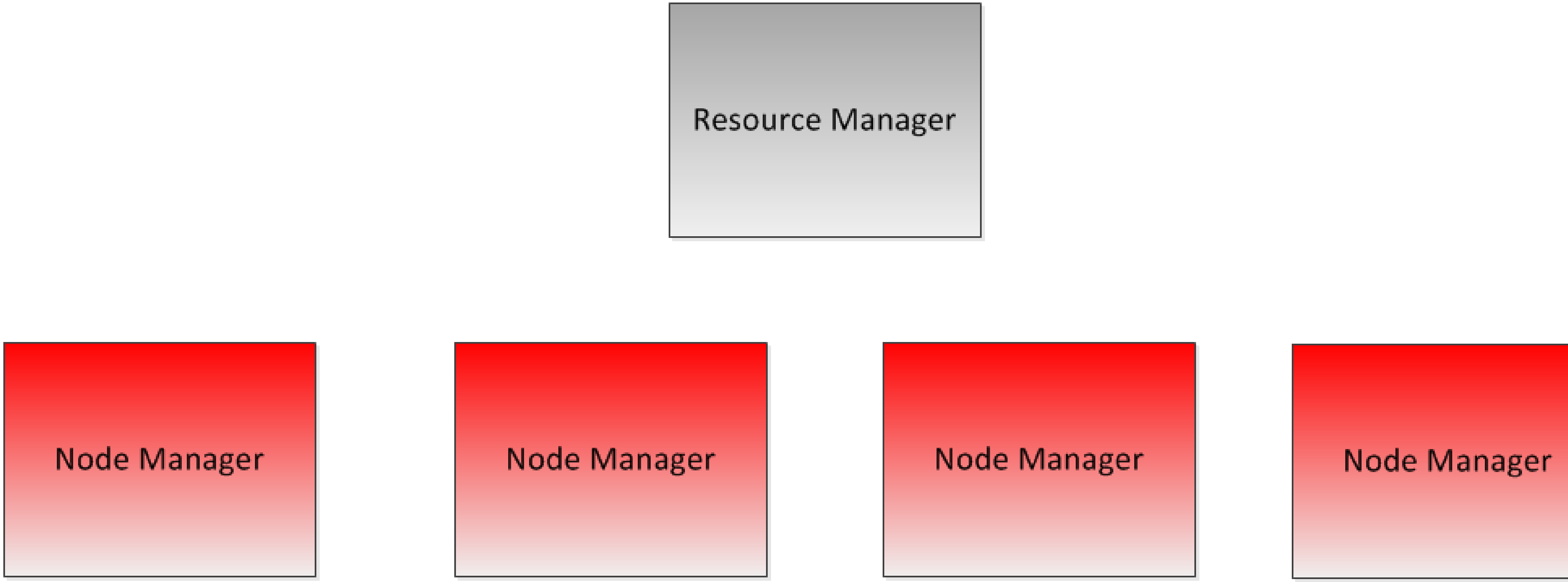
YARN

- YARN coordinates tasks running across the cluster.
- YARN keeps track of all tasks and reallocates them in case of a node failure.
- YARN consists of two components.
- First, the Resource Manager, which runs on a single master node.
- Second the Node Manager, which runs on all the nodes.
- These managers are implemented as UNIX daemon processes.

YARN

•The Resource Manager schedules tasks across all nodes.
•The Node manager keeps track of all individual jobs on that specific node.
•YARN consists of two components.
•First, the Resource Manager, which runs on a single master node.
•Second the Node Manager, which runs on all the nodes.
•Often, the node manager process is co-located with the data nodes.

Yet Another Resource Negotiator (YARN)



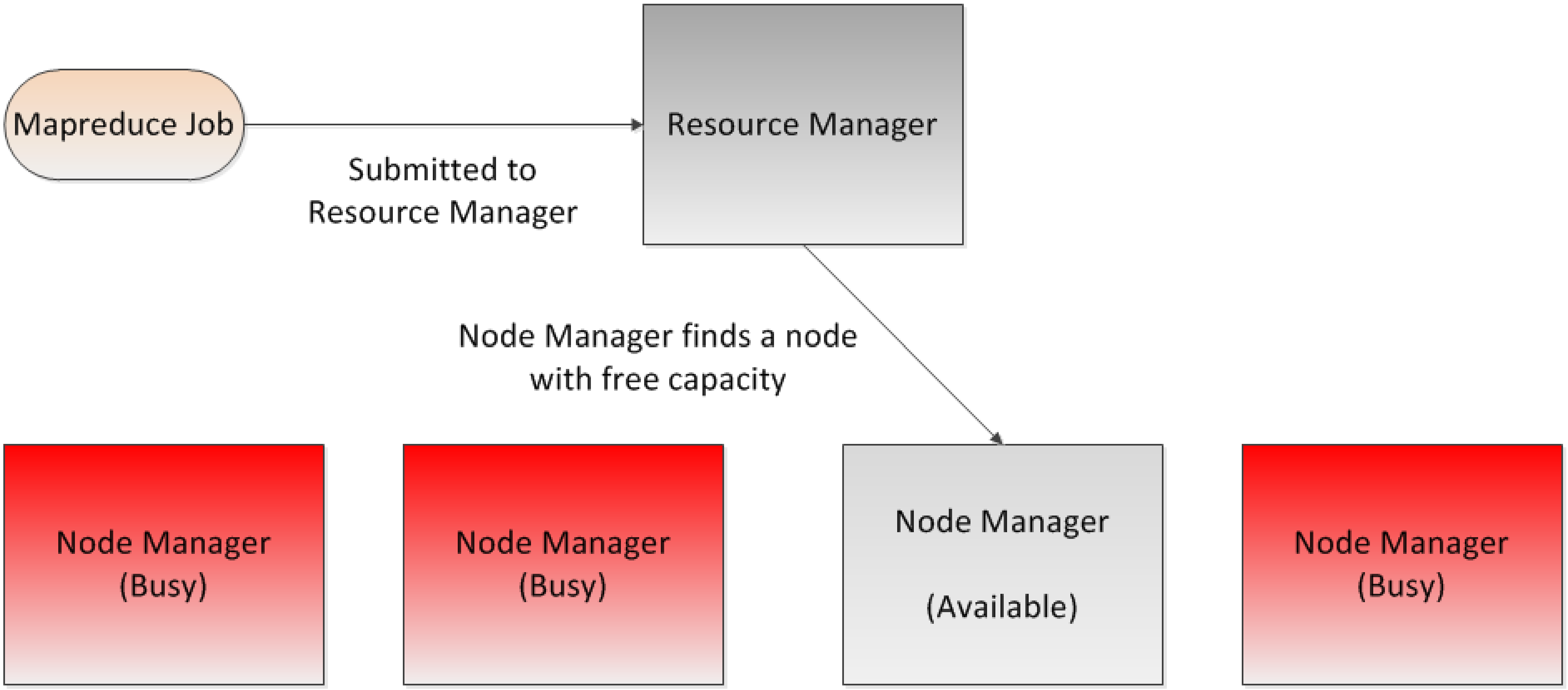
Resource Manager

Node Manager

Node Manager

Node Manager

Node Manager



Mapreduce Job

Submitted to
Resource Manager

Resource Manager

Node Manager finds a node
with free capacity

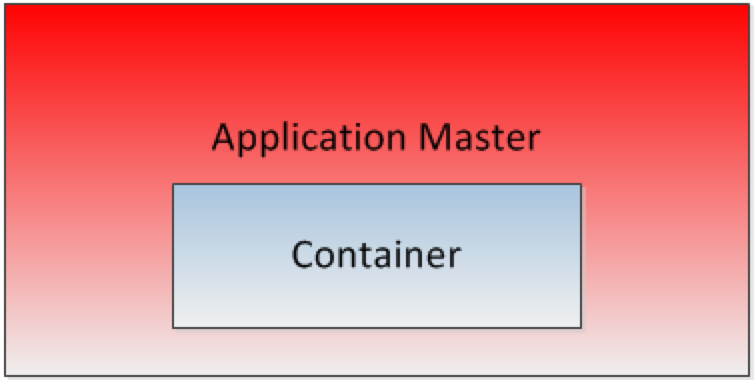
Node Manager
(Busy)

Node Manager
(Busy)

Node Manager

(Available)

Node Manager
(Busy)

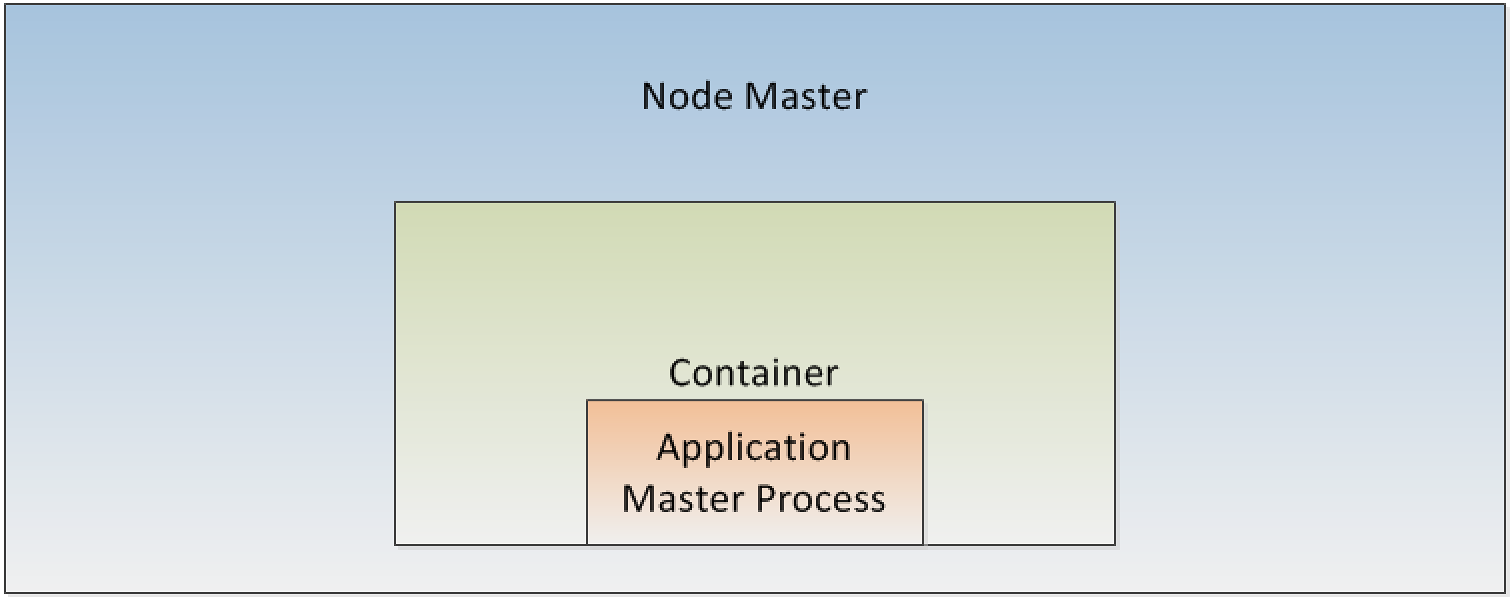


All processes on the node are run inside a container

This is the logical unit containing the resources the process needs, i.e. Memory, CPU, etc.

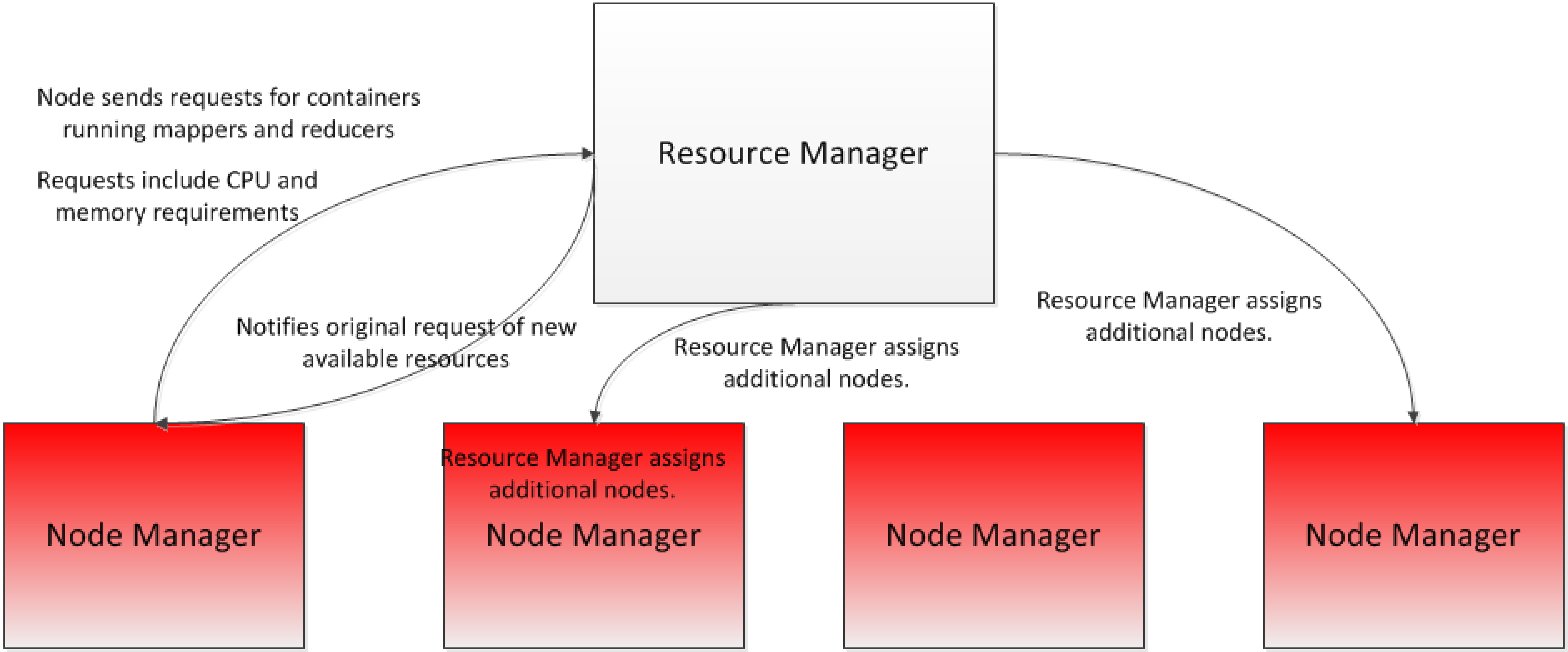
A container runs a specific application.

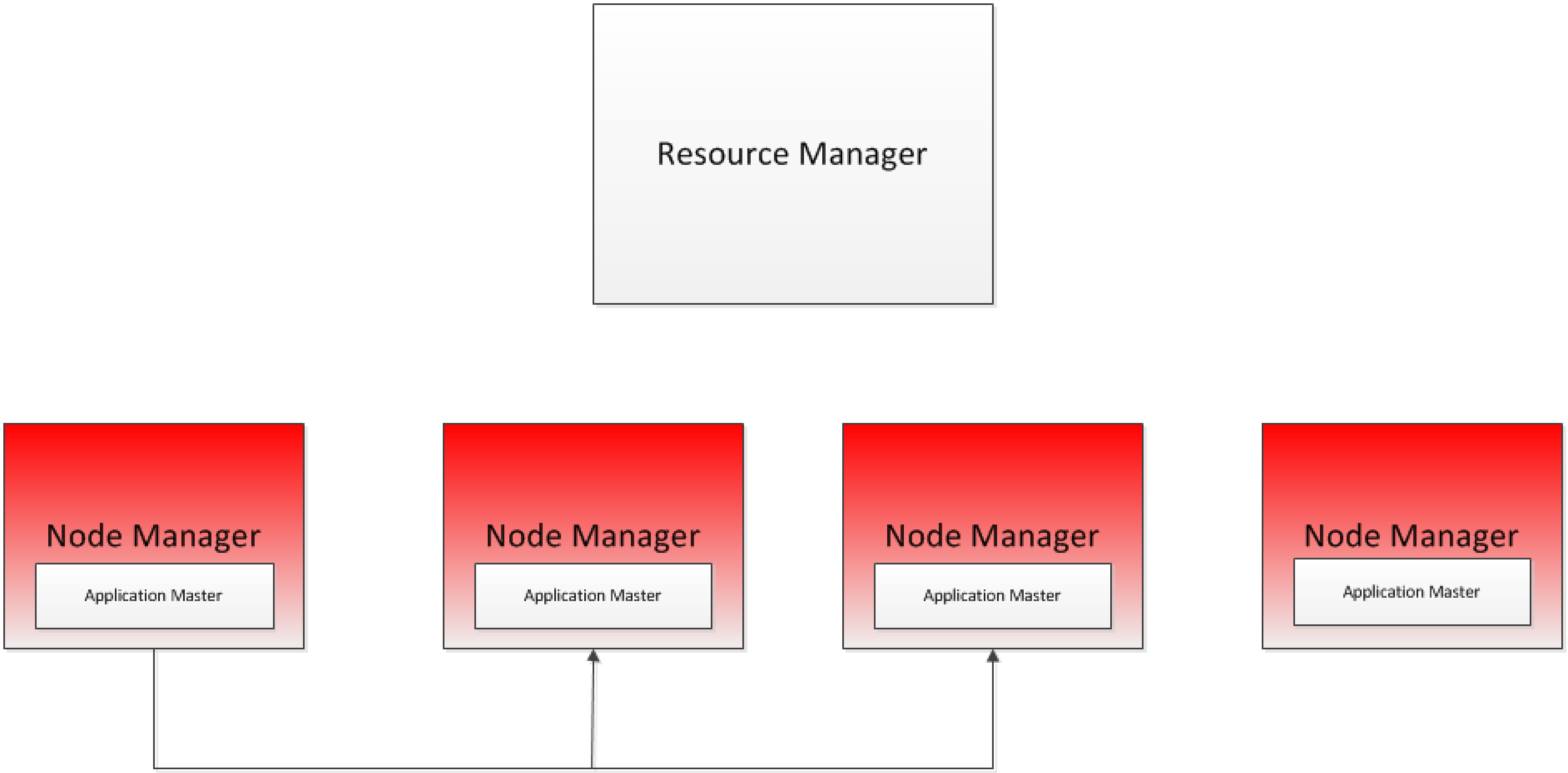
A node can have more than one container.



Performs the computation required for the task

If additional resources are required,
the application master makes the request


Resource Manager
Node sends requests for containers running mappers and reducers
Requests include CPU and memory requirements
Notifies original request of new available resources
Resource Manager assigns additional nodes.
Resource Manager assigns additional nodes.
Resource Manager assigns additional nodes.
Node Manager
Node Manager
Node Manager
Node Manager



Resource Manager

Node Manager

Application Master

Node Manager

Application Master

Node Manager

Application Master

Node Manager

Application Master

Application Master on requesting node starts off application masters on newly assigned nodes.

YARN

- All processes running in a node are run in a container.
- A container is a logical structure.
- A container is described by the resources it owns. For example, memory, CPU, storage, etc.
- A container runs a specific application assigned to it.
- A single node manager can support multiple containers.
- Once the container has been created, the Resource manager starts the application process within that container.

YARN

- If the application requires more resources to complete its task, it makes a request for more containers via the Node manager to the Resource Manager.
- Requests can include CPU or memory.
- The Resource Manager scans the cluster looking for compute nodes with the available resources.
- If it finds a suitable node, it requests the node manager to create a new container and assigns the extra work to it.
- Finally, the original application master making the request is notified of the additional resources.
- The application master than starts up new application masters on the newly available nodes.

YARN

- Most often, clusters have limited resources, so YARN will have to schedule work according to resource availability.
- When scheduling work, several considerations must be taken into account.
- First, write bandwidth must be limited.
- It is better to assign the map computation close to where the data lives.
- If the optimal node is busy, then we wait for the resources to become free.
- We do this using scheduling.
- Hadoop, by default, uses three types of scheduling algorithms.

YARN

- The first type of scheduling is the *First In, First Out* (FIFO) scheduler.
- FIFO scheduling simply means that the first job submitted will take priority.
- FIFO scheduling can be inefficient, if Job A is submitted first, and takes significantly more time to complete than Job B, that means that we cannot run Job B first.
- FIFO scheduling can results in long wait times for jobs, therefore it is seldom used in practice.

YARN

- The next type of scheduling algorithm is the *capacity* scheduler.
- The capacity scheduler implements multiple queues
- Each queue is allocated a portion of the cluster's resources.
- Jobs can be submitted to specific queues.
- Within the queue itself, FIFO scheduling is used.
- Note that queues cannot poach each other's allocated resources.
- Capacity scheduling allows for parallelization of tasks.
- The advantage of capacity scheduling is that small jobs don't get stuck behind larger ones.
- The disadvantage is that you may have underutilization of some of the queues.

YARN

- The last type of scheduler is the *fair* scheduler.
- In a fair scheduler, resources are always allocated proportionally to each job.
- There is no wait time for any job.
- The default policy for Hadoop is the capacity scheduler.
- The default policy can be set in yarn-policy.xml

Apache Pig

- Apache Pig is a dataflow language for expressing data analysis and infrastructure processes
- Pig uses both HDFS (to do file I/O) and MapReduce (to run jobs)
- Pig is extensible through user defined functions that can be written in Java or other languages.

Apache Pig

- Pig has three components
- The first is the Pig Latin language.
- Pig Latin is a high level scripting language
- Pig Latin doesn’t need any metadata or schemas
- All PigLatin statements translated into a series of MapReduce jobs.

Apache Pig

- Pig has three components.
- The second component is Grunt.
- Grunt is the interactive shell for Pig.

Apache Pig

- The Pig Latin language is a language for expressing data analytics and infrastructure processes.
- Supports many types of traditional data operations, such as join, sort, filter and others.
- Simplifies joining data and chaining jobs together

Apache Pig

- Hcatalog can be used with Pig to offload location and metadata information requirements from Pig.
- We no longer need Pig scripts to handle the data locations.
- The dataflow model with Hcatalog looks like this:
- Load (Hcatalog)
- Transform (Pig)
- Dump or Store (Pig)

Apache Pig

- Pig Latin Scripts describe a Directed Acyclic Graph Where the edges are dataflow and the nodes are data operations.

Apache Pig

```
grunt> LOGS = load ‘sample.log’
grunt> LEVELS = FOREACH  LOGS GENERATE (
                          REGEX_EXTRACT($0,’(INFO|DEBUG|WARN|ERROR|FATAL)’,1) AS
LOGLEVELS
grunt> DUMP LOGLEVEL
```

Apache Pig

•The Pig interpreter evaluates each statement.
•If the statement is valid, Pig adds it to the 'Plan' which is built by the interpreter..
•The interpreter will not execute any statements until it finds a DUMP or STORE command.

Apache Pig

- Pig uses the concept of a ‘bag’ as its relational structure.
- A bag is a collection of unordered tuples.
- Each tuple corresponds to a row in a relational table.
- Unlike a standard SQL relation, tuples don’t have to have the same data types or sizes

Apache Pig

- Pig has three data structures
- The first is the bag
- The second is the tuple
- The third is the map (a collection of key/value pairs).

Apache Pig

```
logevents = LOAD ‘input/mylog’ AS (date:chararray, level:chararray,code:int,
message:chararray)
Severe = FILTER logevents BY (level = ‘severe’ and code >=500
Grouped = GROUP severe BY code
STORE grouped INTO ‘output/severeevents’
```

Apache Pig

```
e1 = LOAD ‘pig/input/File1’ USING PigStorage(‘,’) AS (name:chararray,age:int,l
                                                      zip:int, salary:double)
f = FOREACH e1 GENERATE age,salary;
DESCRIBE f;
DUMP f;
```

Apache Pig

```
Employees = LOAD  ;pig/input/File1’ USING PigStorage(‘,’)
                    AS(name:chararray,age:int, zip:int, salary:double);
sorted = ORDER employees by Salary;

Employees = LOAD ‘pig/input/File1’ USING PigStorage(‘,’)
                    AS (name:charray, age:int, zip:int, salary:double);
Agegroup = GROUP employees by age;
li = LIMIT agegroup 100;
```

Apache Pig

```
e1 = LOAD ‘pig/input/File1’ USING PigStorage(‘,’)
        AS (name:chararray,age:int,zip:int,salary:double)
e2 = LOAD ‘pig/input/File2’ USING PigStorage(‘,’)
       AS (name:chararray, phone:chararray);
e3 = JOIN e1 by name, e2 by name
DESCRIBE e3;
DUMP e3;
```

Apache Pig

- We can use illustrate, explain, and describe to help us debug pig latin scripts
- Suggest using local mode to test the script before running it on the cluster
- Local node is slow, but there is no waiting for a slot for the job to run
- Logs for local mode appear on your screen instead of on the remote task node.
- Local mode runs all in your local process so you can attach a debugger to it.

Apache Hive

- Hive is a data warehouse layer build on top of Hadoop
- Allows you define a structure for your unstructured Big Data
- Simplifies analysis and queries with an SQL like language called HiveSQL

Apache Hive

- Hive is not a relational database
- Hive uses a database to store metadata but all data that Hive processes is stored in the HDFS.
- Hive is not designed for OLTP
- Hive runs on Hadoop (a batch processing system)
- Hive latency is generally high
- Hive is not suited for real-time queries and row-level updates.

Apache Hive

- Hive takes multi structured data and gives back a view and a structure which can make sense to a business analyst.
- Hive supports uses such as
- Ad-hoc queries
- Summarization
- Data analysis

Apache Hive

- The Hive architecture includes :
- The Hive command line interface
- The Metastore, which stores schema information and provides structure to stored data
- The Hive QL, which supports query processing, compiling and optimizing

Apache Hive

- Hive is a good choice when you want to query the data
- When you need an answer to a specific question
- If you are already familiar with SQL
- Pig is a good choice when you want to perform Extract , Transform and Load (ETL)
- When you want to prepare data for easier analysis
- When you have a long series of steps to do.

Apache Hive

- The HiveQL is similar to other SQLs
- It uses standard relational database concepts such as tables, rows, columns and schema)
- Supports multi-table insert via your code. It accesses 'Big Data' via tables
- Converts SQL queries into MapReduce jobs
- Also supports plugging custom MapReduce scripts into queries

Apache Hive

- A Hive table consists of:
- Data: typically a file or group of files stored on the HDFS.
- Schema: in the form of metadata stored in a relational database
- Schema and data are separate
- A schema can be defined for existing data
- Data can be added or removed independently.
- Hive can be pointed at existing data.
- You must have a schema defined if you want to point Hive to existing data. In the HDFS.

Apache Hive

```
Hive> CREATE TABLE mytable (name string, age int)
        ROW FORMAT DELIMITED
        FIELDS TERMINATED BY ‘;’
        STORED AS TEXTFILE;
```

Apache Hive

```
Hive> LOAD DATA LOCAL INPATH
        ‘/tmp/customers.csv’ OVERWRITE INTO TABLE customers;

Hive> LOAD DATA INPATH
        ‘user/train/customers.csv’ OVERWRITE INTO TABLE customers

INSERT INTO birthdays SELECT firstName, lastName, birthday FROM
                            customers where birthday IS NOT NULL;
```