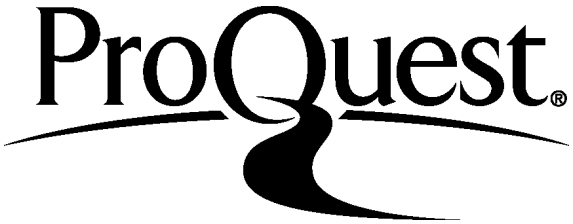
AUTOMATIC DEPLOY HADOOP CLUSER ON AMAZON ELASTIC COMPUTE CLOUD

by

Hao Chen

A thesis submitted to the Faculty of the University of Delaware in partial fulfillment of the requirements for the degree of Master of Science in Electrical and Computer Engineering

Fall 2015



AUTOMATIC DEPLOY HADOOP CLUSER ON

AMAZON ELASTIC COMPUTE CLOUD

by

Hao Chen

Approved: _____

Stephan Bohacek, Ph.D.
Professor in charge of thesis on behalf of the Advisory Committee

Approved: _____

Kenneth E. Barner, Ph.D.
Chair of the Department of Electrical and Computer Engineering

Approved: _____

Babatunde Ogunnaike, Ph.D.
Dean of the College of Engineering

Approved: _____

Ann L. Ardis, Ph.D.
Interim Vice Provost for Graduate and Professional Education

ACKNOWLEDGMENTS

My sincere regards to my supervisor Dr.Stephan Bohacek for his guidance and support throughout the process of developing this project and equipping me with adequate knowledge and skills to do my thesis. It would not be possible to write this thesis without his help. I have enjoyed in the two years I have worked with him. I also want to thank my friend Yongwei Ma for his extremely helpful suggestions to my paper.

TABLE OF CONTENTS

LIST OF FIGURES

ABSTRACT

With the explosive amount of data generated everyday, Big-Data is becoming one of the most popular topics today which receives both research and business attention. Hadoop, which was built based on the Google proposed algorithm MpaReduce, was first introduced by Doug Cutting and his group in 2005. Then it became an Apache project in 2008 and its improved second version was released in 2012. Hadoop has dominated the Big-Data framework area that it is considered as the first choice for most companies and research groups. To deploy a Hadoop cluster, we need to build a computer cluster with a number of nodes. It might not be affordable for small business and research groups with limited funding, so the cloud computing service becomes the alternative. AWS(Amazon Web Service) is one of the most popular cloud providers due to their high quality service and more affordable price. However, building a Hadoop cluster on AWS manually is time consuming due to various factors. Firstly, node information is not fixed when a new set of nodes is requested. There is also extra work to log into all instances to edit their configuratons, it becomes worse when the cluster size is over hunderds or throusands. To address this impeding, this paper has proposed a method to automatically deploy an any size of Hadoop cluster on AWS, including installation and configuration. This saved time which can be spent on more important work.

Chapter 1

INTRODUCTION

Big-Data is a popular buzz-word lately, both in scholarly work and in business. It relates to the explosive growth of data. Big-Data, including any unstructured data, semi-structured data or structured data was characterized with three keywords: **Volume**, **Velocity** and **Variety** by [1] in 2001. Volume describes the large amount data we need to handle,[2] has reported that nearly 2.5 quintillion bytes of data was being created in a single day. Velocity describes the high speed of data generation and the associated information is time-sensitive, thus it requires the corresponding quick data analysis. Variety shows the data are existing with all kinds of different formats, like the traditional structured data in database, or unstructured data like video , audio, pictures and etc. In 2015, three new words were introduced by [3] to better describe features of the term Big-Data. They are **Variability**, **Veracity** and **Complexity**. Variability describes the inconsistency of the data, the same type of data may have different meaning at same time but in different circumstance. Veracity describes the quality of the source data. Noise or bias in data may significantly change the outcome of the analysis. Complexity shows the difficulty of data management. Over the years, Big-Data has extremely grown so much that traditional data management tools, such as relational database is not able to efficiently handle it. Thus in order to explore the true value in the data, it is crucial to have advanced analytic technique and powerful management tool.

Traditionally, data was stored and processed in a database, like Oracle database or MS-SQL server. However, these standarded databases were not capable of handling high volumes of data.of solving such issues, several programming models are proposed.

In 2000, ECL(data-centric programming language)[4], a C++ based programming language, was introduced by Seisint Inc. With ECL, the client is able to declare the data-flow and create data query at a very high level and thus no need to consider the low level system code. In 2004, Google proposed an algorithm called MapReduce which divides a task into many parts and sends them to different computers inside one cluster, and generates the final result from the sub-results sent back from these computers. MapReduce is probably one of the most successful programming models for big data, and Hadoop is the implementation framework of the MapReduce algorithm.

Hadoop is an open source software which is written by JAVA and runs programs based on MapReduce algorithm. It was first introduced to the world by a group from Yahoo led by Doug Cutting. The name Hadoop is inspired by his son‘s toy elephant. Later, Apache Software Foundation took over this project in 2008. In 2013, the MapReduce 2.0 was launched with great improvement on the scalability(the 1.0 version was reported limited at 4000 nodes per cluster). Hadoop is one of the most popular frameworks for Big-Data, not only big companies[5], many small businesses are using Hadoop. It also receives a lot of research attentions[6][7][8][9]. [10] proposed a GPU-Hadoop integrated framework. [11] introduced an improved hybrid scheduler with fair scheduling, FIFO and COSHH to improve Hadoops performance. [12] performs an evaluation on the read and write operations on HDFS system and results show that the file is better to be smaller than a block size to achieve best performance.

Although Hadoop is popular in both business and scientific field, the cost to build a hadoop cluster, especially a large cluster over hundreds of nodes, is an issue to small business and researchers with limited funding. Thus building a Hadoop cluster on cloud computing service will be a good choice[13]. And AWS(Amazon Web Service) is one of the best service providers due to its high quality service and affordable price. However, it is not easy to set up a Hadoop cluster on AWS, because 1)node information is not fixed when we request a new set of nodes 2)we need to log into all instances to edit their configuratons, it becomes worse when the cluster size is over hunderds or throusands. It is desirable to have a tool which can automatically set up all the

nodes, install and configure Hadoop on AWS. In this goal, we analyze the installation procedure and provide a solution that helps people easily set up a working Hadoop cluster.

The rest of this paper is organized as follows: Chaper 2 introduces the history of Hadoop, the architecture of Hadoop Mrv1 and the improved version Mrv2 and compares it to Apache Spark. Chaper 3 highlights the basic steps of setting up a Hadoop cluster manually, then analyzes these steps and provides solution on how to perform these steps automatically. Chapter 4 concludes this paper.

Chapter 2

BIG-DATA PLATFORM: HADOOP

The Apache Hadoop with the MapReduce function is the current mainstay to process the distributed data. It was spearheaded by the Google MapReduce framework and Google file system[14]. Based on its unique scale-out and fine-grained processing infrastructure, the Hadoop received its explosive growth in Big Data area. Furthermore, the application ecosystem is also rapidly developed, such as Apache Zookeeper, Apache HBase, Apache Hive, Impala, Apache Phoenix, Apache Oozie, Apache Pig, Apache Sqoop, Apache Flumeand among others.

2.1 Basics of Hadoop

Hadoop cluster can be running on a single node (this node represents both the master node and date nodes) to extend to thousands of nodes (Different nodes have their own jobs and run in parallel) with very high level fault tolerance guaranteed. 2.1 shows one basic Hadoop cluster.

Generally, one Hadoop cluster can be decomposed into two abstractions: a distributed file system HDFS(Hadoop Distributed File System) and a MapReduce engine.

2.1.1 HDFS

The HDFS provides a storage model. When all of your data is loaded on Hadoop, it is split into several pieces and stored across your cluster. It is able to copy and process data between different nodes, and it supports large files. HDFS has the directory of all the files in its system in order to keep track the location of each data. Furthermore, for each piece of data, there are multiple copies kept, even if one node dies, the lost data

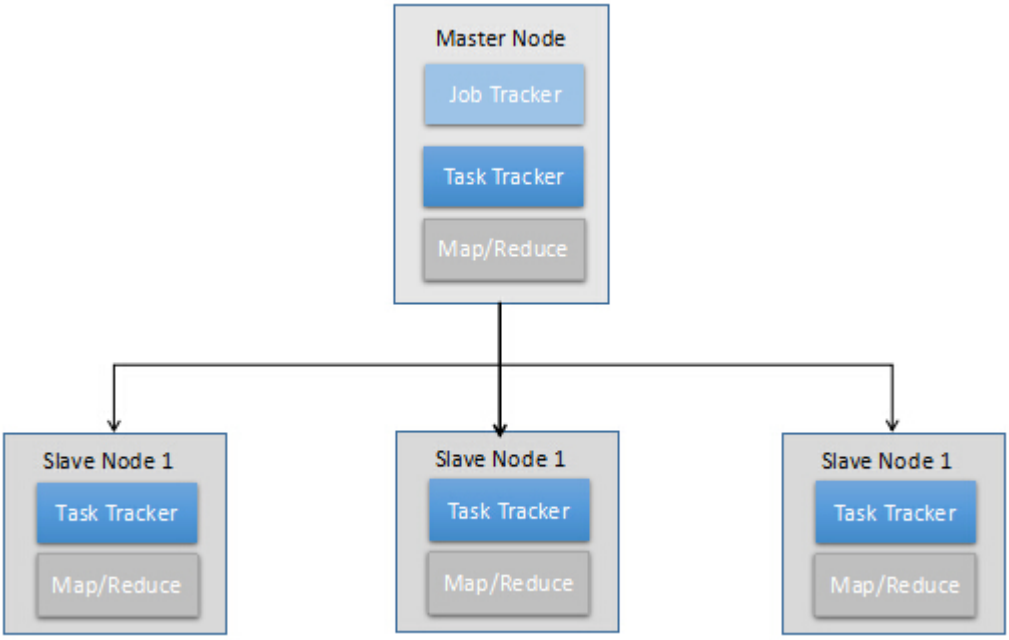


Figure 2.1: MRV1 Architechture

can still be replicated from another location, this is the basis of the fault tolerance of HDFS.

2.1.2 MapReduce Engine

What is MapReduce? Basically, MapReduce means working on a way that parallel executes a program on a cluster. Lets first take a look at what parallel programming is. Parallel programming, as the name implies, is a programming model that a task is divided into several parts and executes these parts at the same time. Most importantly, there should be no dependency between these parts. It means that the input of one part should not be related to any other parts‘ execution.

Compared to simple parallel programming, MapReduce is a more advanced scheme. Usually, its input is a very huge set of data. Because the input is too large to be processed by a single node, and due to time constrains(for example, a Google search result for more than 10 seconds is not acceptable), we need to process these data over thousands of machines simultaneously. Thousands of machines are performing the same computation but with different data input, furthermore, besides fast response time, MapReduce also takes load balancing and fault tolerance into consideration which will be discussed later. In Hadoop, the MapReduce engine consists of two core functions:

1)Map: a procedure which decomposes large input data into two or more small chunks to be processed.

2)Reduce: a procedure which collects all sub-results and generates the final result.

2.2 Two Generations of Hadoop

Hadoop was first created by Yahoo! (Doug Cutting‘s team) in 2006, and taken over by Apache Software Foundation in 2008. In Dec 2011, the version 1.0 was released in the public domain and two years later, the second generation was released.

2.2.1 First Generation of Hadoop: MRV1

The first generation contains two major components: Job Tracker and Task Trackers. Job Tracker runs on NameNode(the master node in Hadoop) and plays like the central coordinator of this cluster. It is responsible for allocating jobs to Task Trackers, manage all the resources on the cluster and keep track of all the running tasks. Each Task Tracker runs on one DataNode(the slave nodes in Hadoop) in the cluster and is responsible for managing the tasks arranged by Job Tracker and keeping contacting Job Tracker to report the current status of its tasks.

Now, lets systematically examine the procedure of executing a MapReduce job. When a client sends a job request to the Hadoop cluster, this request is first received by Job Tracker. This request will contain information like: Map and Reduce functions, the path to the required data, etc.

After receiving the request, the Job Tracker will put the request into the request queue which is executed in FIFO(first in first out) manner.

When executing a request, the Job Tracker will first decide how to decompose this request (the number of splits). Then the Job Tracker will communicate to HDFS to get the location information of the required data. After gathering the data location information, the Job Tracker will allocate tasks to the DataNode which has available slots (the number of slots means the number of tasks this node can execute in parallel) and shortest distance to the required processing data. When a Task Tracker receives a job, it starts to prepare the necessary files for this job, such as the Jar file. Concurrently, the Task Tracker keeps reporting the status of all jobs running on this node to the Job Tracker. When the Job Tracker finds that all Map tasks are done, it will notify the selected Task Tracker to execute the corresponding Reduce jobs. When all work are done, the Job Tracker will update the status and notify the client the job is finished.

When executing the tasks, two separate Java virtual machines will be launched by Task Tracker to prevent the failure which may be caused by current running jobs.

The Task Tracker will keep sending a heartbeat signal to Job Tracker every few minutes to indicate this Task Tracker is still alive. If the Job Tracker does not sense

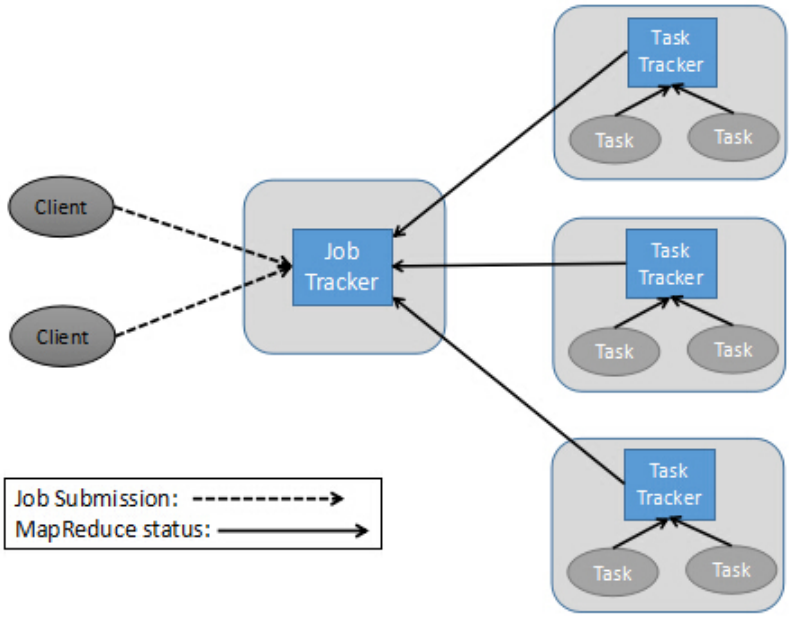


Figure 2.2: MRV1 Task Flow

the heartbeat signal for a certain time, the Job Tracker will consider this Task Tracker has failed and will re-assign its jobs to another available Task Tracker.

2.2.2 Second Generation of Hadoop: MRV2

Compared to MRV1, MRV2 add a completely new layer into its framework, it is call YARN(Yet Another Resource Negotiator), which is located in mid of HDFS and MapReduce engine. The functionality of YARN is to manage all the resource on this cluster, such as memory, CPU (More resource will be added in future version). So MapReduce engine now is focusing on performing data processing function only.

2.2.2.1 Issues of MRV1

In MRV1, the resource management is controlled by a component called Job Tracker which is part of the MapReduce engine. And Job Tracker also monitors each scheduled task. If it finds any failed job, it re-allocates a new job to a new DataNode.

MapReduce jobs consist of Mappers(Map jobs) and Reducers(Reduce jobs). Mapper and Reducer can be running on the map slot and reduce slot on DataNode respectively. Although MRV1 is one of the most successful schemes in the Big-Data area, it still has many issues which need to be fixed:

1) Scalability: In MRV1, there's only one Job Tracker per cluster. So it will be the bottleneck when the cluster size grows up. It is determined that the maximum number of nodes on a MRV1 cluster is only 4000 which is not able to meet the needs of some large companies nowadays.

2) Availability: Apparently, the Job Tracker is the single point of failure of MRV1. If the Job Tracker fails, all jobs need to be restarted.

3) Resource utilization: In MRV1, Mapper can only be executed on the Map slot on DataNode and Reducer can only be executed on the Reduce slot on DataNode. In certain times, the amount of Mapper and Reducer are unbalanced. This happens when the Map slot is full but the Reduce slot is empty. Hence, the computer resource (memory, CPU) is not fully utilized to maximize the speed of computation.

4) No-MapReduce job support: the MapReduce programming model is great but it is not good for every application. Some other programming models, such as MPI, real-time processing and graphics processing is not easy to be converted to MapReduce model. In MRV1, Job Tracker is specifically designed to schedule and monitor MapReduce jobs. So it is essential for Hadoop to support these models that the company that has the Hadoop-based cluster can also easily deploy such applications on their cluster to lower the cost. Otherwise they need to build a new cluster and move data back and forth.

2.2.2.2 Sencond Generation of Hadoop: MRV2

In order to overcome the issues mentioned in last section, MRV2 removed Job Tracker and Task Tracker from MapReduce engine. Instead, a new design YARN is placed as a new layer between the HDFS and MapReduce engine to take over the job of resource management. All applications (including MapReduce one and normal one)

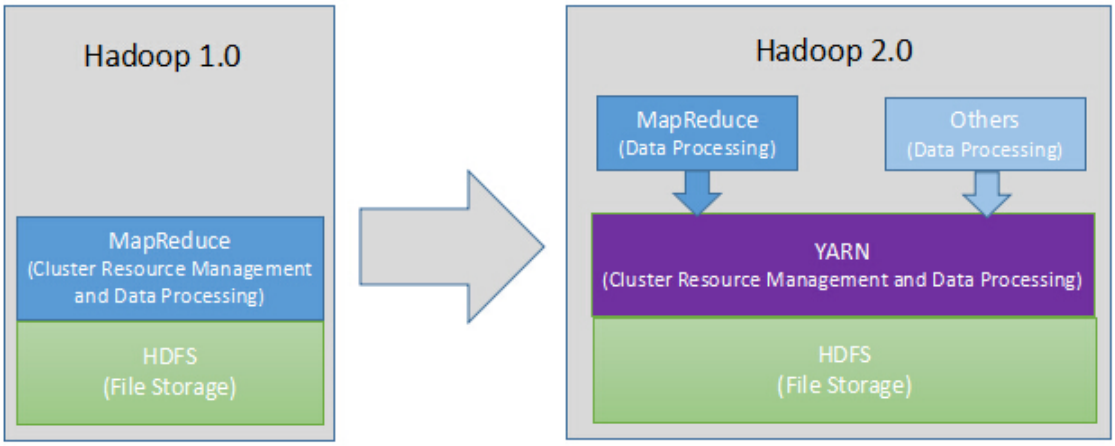


Figure 2.3: Changes on MRV2

share the same resource management from YARN. Also, the definition of Map-slot and Reduce-slot is removed. Every task will have a certain amount of resources, such as memory and CPU cores. Furthermore, the function of old Job Tracker is replaced by a scheme called Application Master, and we can launch multiple Application Master in one cluster.

Now, let's closely examine MRV2, before we discuss the procedure how MRV2 works, we first need to understand some key concepts in MRV2:

1)Resource Manager: the Resource Manager only runs on master node, it manages and allocates the resources of this cluster to all requested applications through a pluggable global scheduler. The scheduler does not monitor or track any information of the launched application, it only allocates resources based on the applications requirement and several predefined constrains.

2)Container: Container is an abstract concept in MRV2 that represents the amount of available resources (memory, CPU cores) on one DataNode. It is created by Resource Manager upon a successful resource request from an application. Once a Container is allocated, it is controlled by Application Master and used to launch and

execute tasks.

3)Node Manager: there‘s one Node Manger per node. It is responsible for preparing the Container which is allocated by Resource Manager, monitoring these Containers‘ status, such as CPU usage, memory usage, the network condition and also keeping informing Resource Manager with the monitored information mentioned above.

4)Application Master: the Application Master is launched one per application. Its job is to negotiate with Resource manager about the container the application is needed. Container is just a set of resources of one machine in the cluster. With the allocated Container, the Application Master needs to communicate with Node Manger, provides the application specific information to Node Manger to launch the tasks, monitors the status of the Containers (such as resource consumption) and the application‘s progress. Due to the effort of Application Mater, the YARN is able to improve its:

Scalability: The Application Master is designed to have a pure scheduler, so it only focuses on the resource scheduling without considering other functions like fault tolerance. Also the Application Master is launched as one per application, so itself will not be an issue to the clusters scalability. It is shown that, with the Application Master design, the YARN cluster is able to easily scale to over 10000 nodes without any major issues being reported. It is a great improvement compared to the 4000 nodes limitation in MRV1.

Compatibility: With Application Master handling all application-specific code, YARN is able to support more programming models other than MapReuce, such as MPI, graphics processing. Furthermore, YARN is able to deploy any kind of application, in MRV1, only JAVA code is supported.

In YARN, Application Master is designed to be one instance per application, but it is not necessarily be this model. In Apache Pig and Apache Hive(based on YARN), Application Master is designed to support more than one application. With the concepts introduced above, we now can step into the details of the procedure on

how an application is launched on Hadoop-YARN cluster:

1)The client submits the target application to Hadoop-YARN cluster. The information includes the application name, application path, required data to execute the application and application required resource.

2)The Resource Manger first receives the client application launch request. It needs to find a DataNode and allocate a special Container for an Application Master and launch the Application Master on the DataNode .

3)When the Application Master is launched, it will first register itself with the Resource Manager. After this registration, the client is able to directly talk to the Application Master if any information needed.

4)Then the Application Master starts communicating to the Resource Manger to ask for enough resource (containers) to launch this application based on client‘s request.

5)Once the Resource Manager finds enough resource to meet this application‘s request, it notifies the corresponding Node Manger and launches the required amount of Containers on these nodes. The necessary launch information is provided to the Node Manager by the Resource Manager. After this, the Container is able to communicate with the Application Master.

6)The application‘s code is deployed on the Container and the application starts to execute. In the meantime, the Application Master monitors the applications execution status.

7)When the application is executed, the client can directly communicate with the Application Master about the execution status.

8)Once the task finished, the Application Master will report the Resource Manager, turn off itself and free its own Container.

2.3 An Alternative Data Framework:Apache Spark

Although the new Hadoop-YARN framework greatly improves the Hadoop cluster and supports other normal applications deployed on the cluster, it is still designed

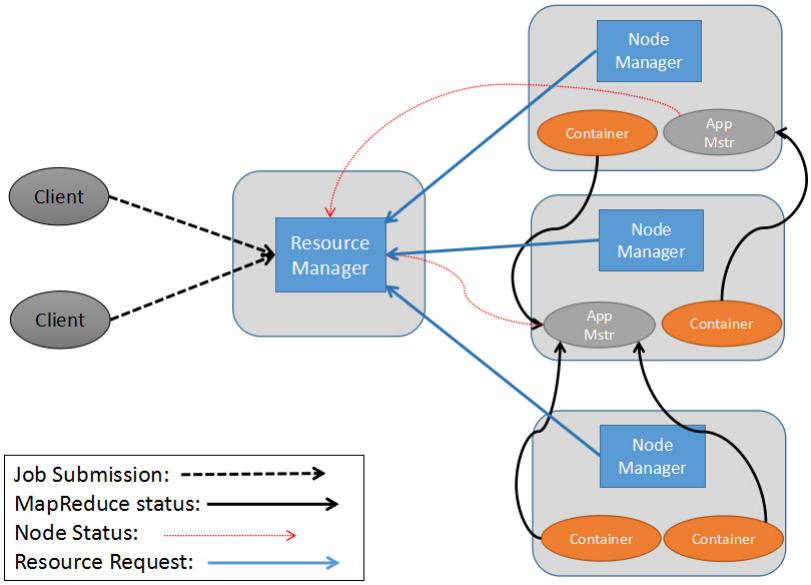


Figure 2.4: MRV2 Task Flow

to mainly support these two-step Map-reduce batch processing tasks. As the rapid development of current Big-Data area, some other frameworks come out to compete with Hadoop MRV2 with some unique design, such as the in-memory processing from Apache Spark, and successfully replace Hadoops dominance in certain areas. In the next part, I will simply introduce the structure of Apache Spark and the differences between Hadoop and Spark.

Apache Spark is an open source Big Data based project which was first introduced by university of California, Berkeley and now managed by the Apache Software Foundation. It is developed with a programming language called Scala, and executes under the JAVA environment, and multiple programming languages are supported on the Spark cluster. These are:

1) SCALA

2) PYTHON

3) JAVA

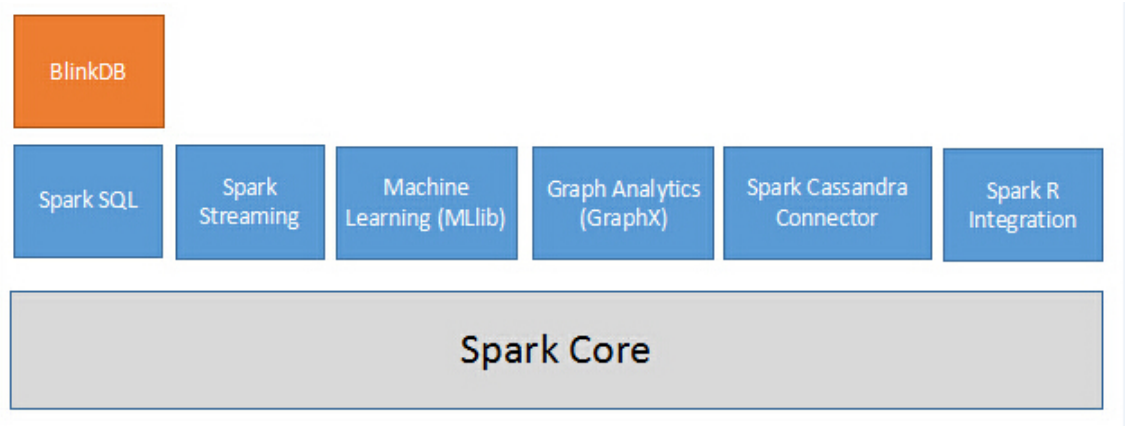


Figure 2.5: Spark FrameWork Ecosystem

4) R

5) CLOJURE

Spark is designed to improve the Hadoop, so the traditional 2-step Map-Reduce model is supported. Besides the Map-Reduce, it also supports other operations like graphics processing(GraphX), machine learning(MLlib), streaming processing and SQL query. Test result shows Spark is able to run 99 times faster than the traditional Hadoop[15] due to its memory based operation, compared to Hadoops disk based operation. Now, let‘s take a little more detailed look into Spark:

Basically, Spark consists of 3 major components: the data storage, API and a cluster resource manager.

1)Data storage: Spark can use the HDFS system, Apache Cassandra, Amazon S3 or even custom solution.

2)API: API is used to support other developer more easily write Spark based applications. Currently, the API is supported with 3 languages: SCALA, JAVA and PYTHON.

3)Cluster Resource Manager: Spark can be deployed as a stand-alone native Spark server or deployed based on other resource management framework like YARN

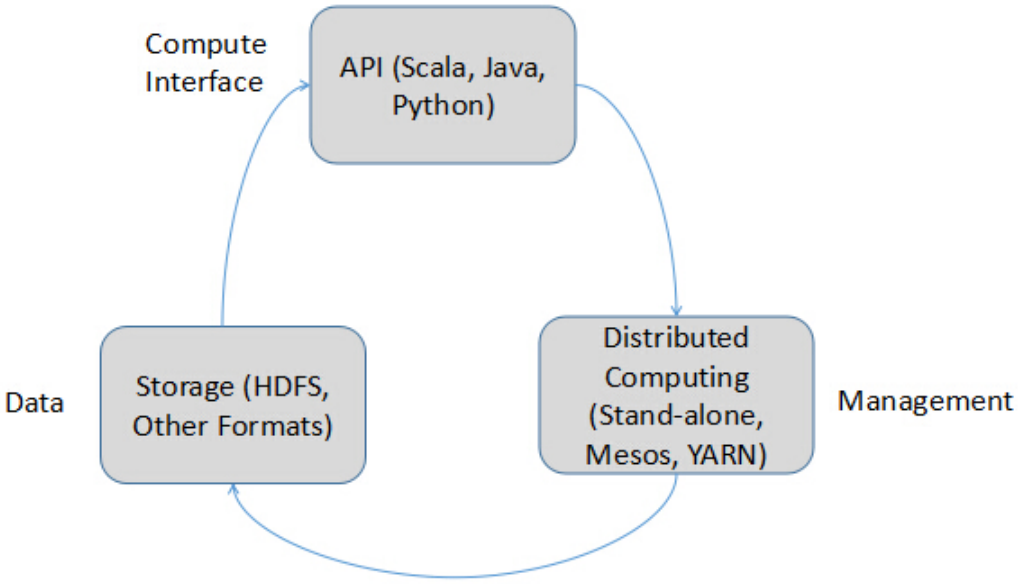


Figure 2.6: Components of Spark

and MESOS.

2.3.1 Resilient Distributed Dataset

Resilient Distributed Dataset is the key feature that Sparks provided to achieve fault-tolerance.

It is Spark‘s fundamental primary data abstraction which can be made and operated in a parallel manner. And it is immutable. There are three ways to create the Resilient Distributed Dataset:s

1)Paralleling an existing collection: if the data is already in Spark, we can operate it in parallel. It is done by calling function called SparkContexts parallelize on a dataset. After executing this function, a new distributed dataset will be generated and it can be used in parallel in future.

2)Referencing a data set which is stored in external system: the dataset can be from any data source: like HDFS, local file system, Amazon S3, etc. It can be done by calling a function called SparkCintext S textFile.

When Resilient Distributed Dataset is created, an associated DAG (directed acyclic graph) is created. To operate these data sets, Spark also provides two methods:

1)Transformation: it is used to update the DAG, but actually nothing really happens until some actions are initiated. Some supported transformation functions include: filter, groupByKey, flatMAP, map, pipe, aggregateByKey, etc.

2)Action: it is able to return a new value. When the function action is called, all the required processing data will be executed and new results will be returned. Some supported transformation functions include: collect, count, reduce, take, first, foreach, etc.

2.3.2 Spark VS Hadoop

Hadoop is a successful Big Data framework. Its key feature is the two-step MapReduce procedure that each operation in this scheme needs to be divided into two phases, Map and Reduce. To fully utilize Hadoop framework, you need to transfer any application into the MapReduce model. All the intermediate and final results from Hadoop are stored in the HDFS system on the cluster disk. In order to achieve fault tolerance, these data are replicated just in case that one of data nodes is down. But the disk reading&writing latency and replication intensively degrades its‘ overall performance. For some complex tasks, you need to connect a series of MapReduce jobs together. And we can not start a new job until the last one is finished.

Spark improves Hadoop by processing the data in memory level and the nearly real-time data processing. Compared to Hadoop which reads&writes the data on disk, Spark holds the intermediate data in memory, this feature is especially useful when this intermediate data needs to be used more than once. The fault tolerance of Spark is achieved by the design of Resilient Distributed Dataset which is able to recover the failure during execution.

However, Spark does not have it own distributed storage system which is one the key feature of Big Data framework. Instead of storing all the data in one huge machine, the distributed storage system is much salable that more data set can be

added later. In order to set up a Spark cluster, it needs to be built on some third party distributed storage system, such as HDFS.

Its important to note that the relation between Hadoop and Spark is not based on competition. Sparks out-performs Hadoop in some areas, such as machine learning algorithm(it is reported it can be 100x faster than Hadoop in some cases). But when we are going to process some traditional batch data processing work, in which case all the data may not be able to fit in to memory, we still need to store data on disk. In this case, Spark is not able to out-perform Hadoop which dominates the Big-Data framework area for the past. Also, Spark is a newly developed tool which may have more bugs which need to be fixed..

Chapter 3

AUTO-DEPLOY HADOOP ON AMAZON EC2 CLUSTER

In the previous section, we have established that Hadoop is important in the Big Data area. Most companies, especially small business with limited funding will choose to deploy their Hadoop clusters on cloud computing service, such as AWS. However, it is not easy to deploy a cluster on AWS because 1)cloud node information is not fixed when we request a new set of nodes 2)we need to log into all instances to edit their configuratons, it becomes worse when the cluster size is over hunderds or thousands. So it is desirable to have a tool which can automatically set up all the nodes, install and configure Hadoop on AWS. In order to implement the function that auto-deploys Hadoop cluster on Amazon Ec2 instances, we need to first understand how this is done in normal case,which means how it is done manually. Based on this, we can then further analyze and discuss how to make this procedure automatic.

3.1 Steps to Build a Hadoop Cluster on AWS Manually

In this case, we will build a Hadoop cluster with 3 Nodes as example, 1 master and 2 slave nodes respectively, within 6 steps.

3.1.1 Create an Amazon Account

The first thing we need to do is to set up an Amazon account at website: https://aws.amazon.com/. By default, one Amazon account is only allowed to run 20 instances at the same time. So one account is good in this case. The registration is pretty easy and straightforward, and I already have one account with Amazon, so I skip this step.

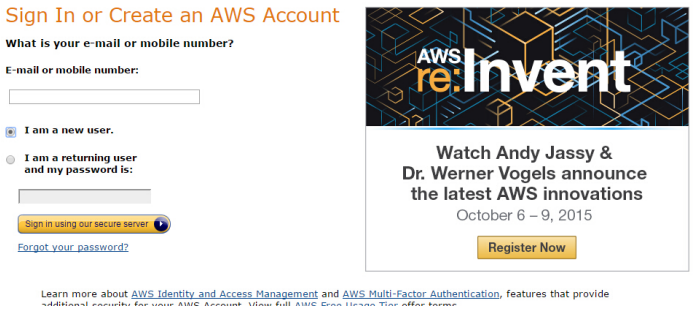


Figure 3.1: Create Account

3.1.2 Launch AWS Instances

Before we launch the AWS instances, we need to do some preparations first:

3.1.2.1 Create a Security Group

The security group is a set of rules which control the behavior of target instances. For example, it can allow access or deny certain types of protocols communication(such as TCP, UDP), also it can stop any access from outside except selected IPs. With the security group, the instance is much more safe to use. The security group button is located in the EC2 control panel under the tag NETWORK & SECURITY. In our case, to build a Hadoop cluster, we need to allow all TCP access, all ICMP access and the SSH access under port 22. I have not limited the access from selected IPs this time, but it can be done to ensure more security.

3.1.2.2 Create a Key Pair

The key pair is used to log into Amazon instance with SSH tools after instances are launched. In EC2, there‘s no user name and corresponding password. The key pair is the only way that provides the identification for you to have the authority to access your instances. To create key pair, the button is also located in the EC2 control panel and under the tag NETWORK & SECURITY. Its straightforward to create a key pair,

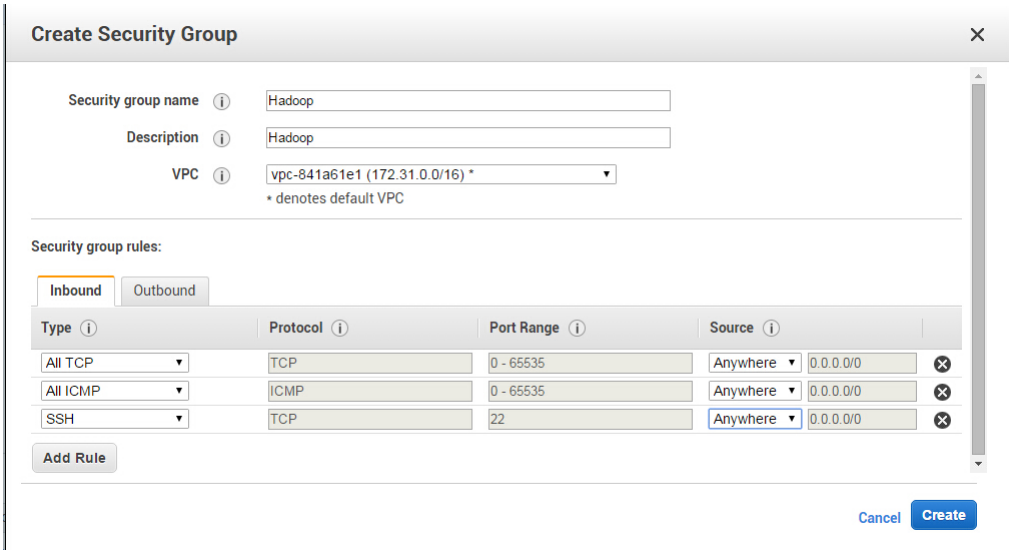


Figure 3.2: Create Security Group

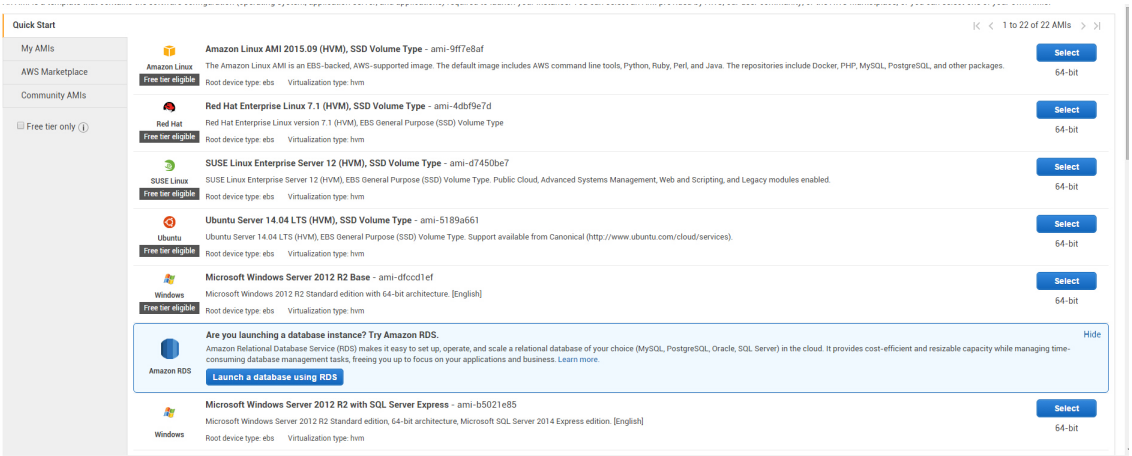


Figure 3.3: Select AMI

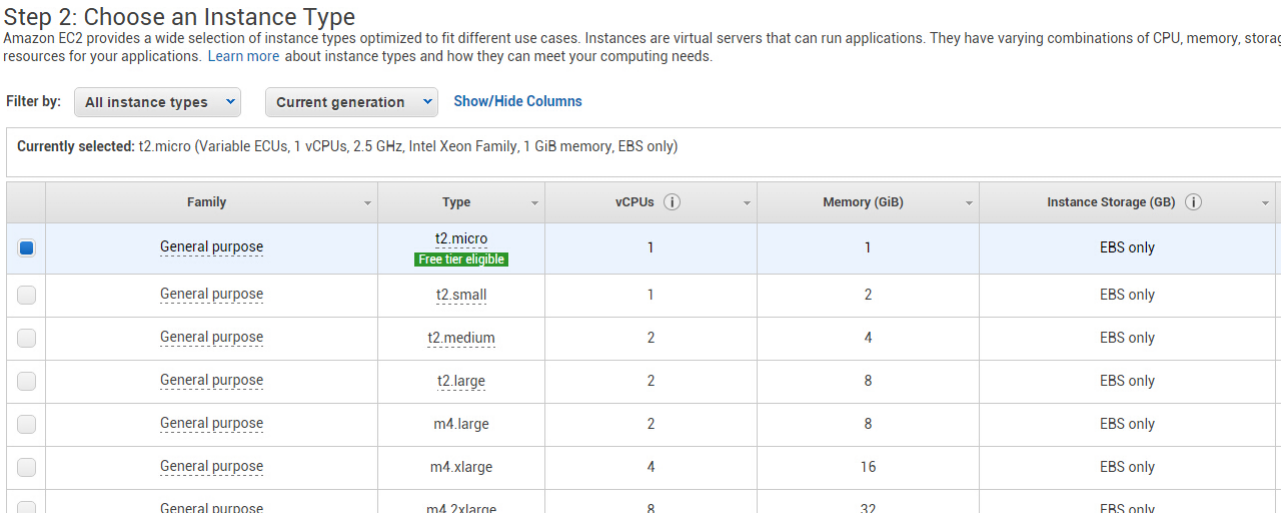


Figure 3.4: Select Instance Type

and a .pem file will be downloaded into your computer. You need to carefully store this key pair as it is your only identification to access the instances.

3.1.2.3 Launch Instance

After setting the security group and key pair ready, we can now launch the AWS instances. To launch an Amazon EC2 instance, we need to firstly select an AMI. AMI, standing for Amazon Machine Image, is an image that the newly launched instance will be. It contains the information including operation system, installed applications and environment parameters, etc. Amazon provides some basic AMIs with basic OS installed (such as Ubuntu, Red hat). With the default AMI, some required applications are pre-installed, such as SSH. But some other necessary applications are not installed. We select the Ubuntu version in this installation. Pick t2.micro in next step, it is free of use for 1 year. Then enter number 3 as the number of instances in next step. We can skip the Add Storage and Tag Instance steps. Then select the security group we just created.

After reviewing all the details to make sure everything is correct, we can click

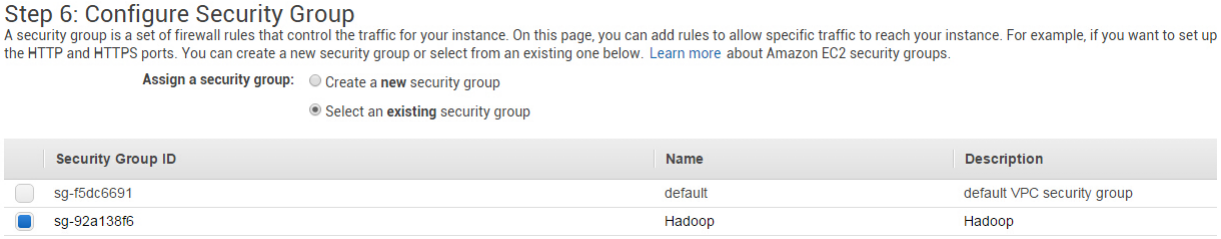


Figure 3.5: Select Security Group

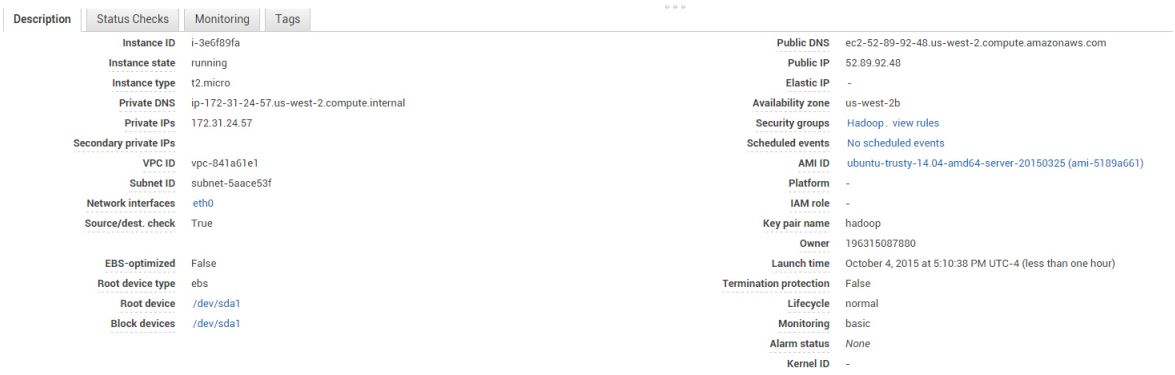


Figure 3.6: Instance Information

the launch button, select an existing key pair as the identification. If it launches successfully, we will have 3 EC2 instances ready to use. The instance information, include the public&private IPs, are located under the tag instances.

3.1.3 Log into Instances

With the public IPs, now we can log into these instances. If you are using Windows machine to log in, you need to prepare two tools: PuTTy and PuTTygen, they can be downloaded at: http://www.chiark.greenend.org.uk/~sgtatham/putty/. If your are not using the Windows machine, please refer the AWS documentation for help.

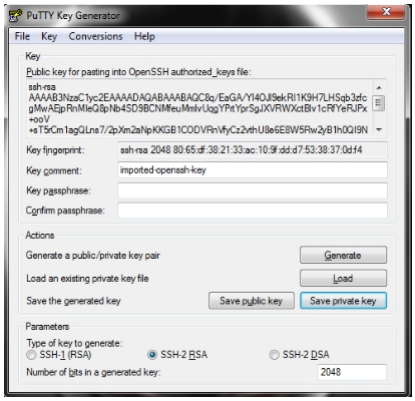


Figure 3.7: Generate PPK File

3.1.3.1 Generate PPK File

Remember we get a .pem key pair in section 3.1.2.2 as our identification, however, puTTy does not support .gem file. So we need to first convert the .pem file into the .ppk file which is supported by PuTTy.

PuTTYgen is used to generate the .ppk file. First select the SSH-2 RSA as the target type and then load the .pem file we get in section 3.1.2.2. After clicking the save private key button, we will get our .ppk file.

3.1.3.2 Log into Instances

Now we have the public IP and required .ppk file, we can use PuTTy to log into Amazon Ec2 instances. Enter your user-name@your-public-ip as the host name(we launched an Ubuntu machine, ubuntu is the user name, if you are using other Linux OS, please use the corresponding user name). For the key pair, click Connection → SSH → Auth, browser and select the correct .ppk file. If everything is set correctly, you will be able to see the similar display as 3.10.

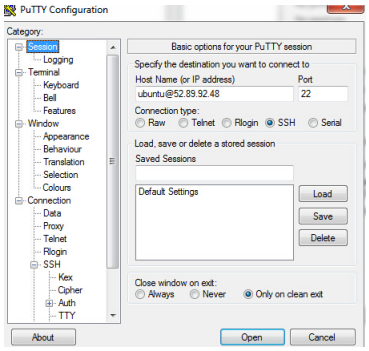


Figure 3.8: Login with PuTTY

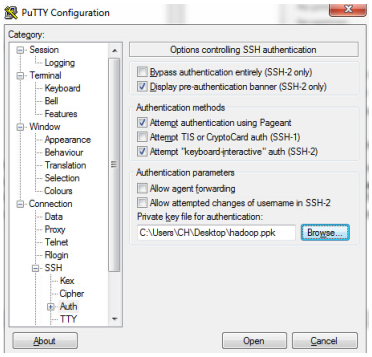


Figure 3.9: Select Key Pair

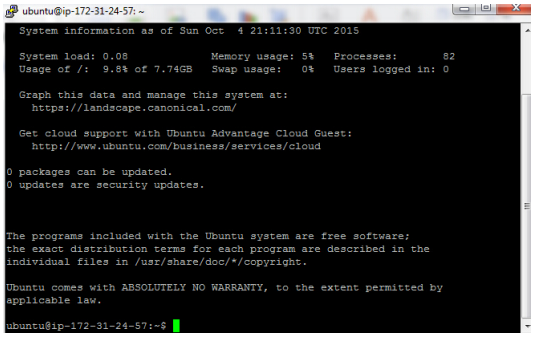


Figure 3.10: Successfully Login

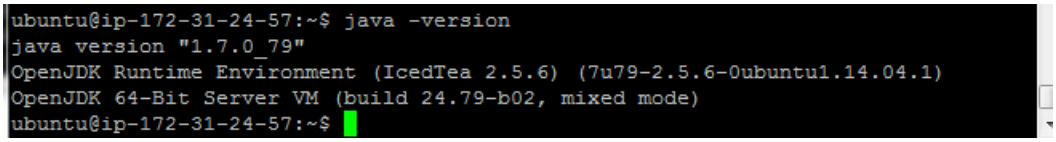


Figure 3.11: Successfully Intalled JAVA

3.1.4 Set up Hadoop Cluster

Now we are successfully logging to the instance. It is just a plain Linux OS and only a few basic tools installed. In order to set up the Hadoop cluster, we need to do the following things:

3.1.4.1 Install JAVA

Before we install JAVA, I suggest to update the apt-get to make sure everything is up to date. To do this, run command:

sudo apt-get update

To install JAVA, we can choose either Oracle JDK or OpenJDK. In this case, I picked OpenJDK 7. To do this, run command:

sudo apt-get install openjdk-7-jre openjdk-7-jdk

When the installation finished, you can run command to test:

java -version

If you see the same message as 3.11, it confirms that you just successfully installed JAVA. After installation, we need to set the correct environment variables, JAVA_HOME to help other applications find the path to JAVA. It can be done by editing the .bashrc file under your default directory. To do this, run command:

vi ~/.bashrc

Add the line export ***JAVA_HOME=/usr/lib/jvm/java-7-openjdk-amd64*** at the first line of this file and save this file. To make this update effective, we need to source this file by running command:

source ~/.bashrc

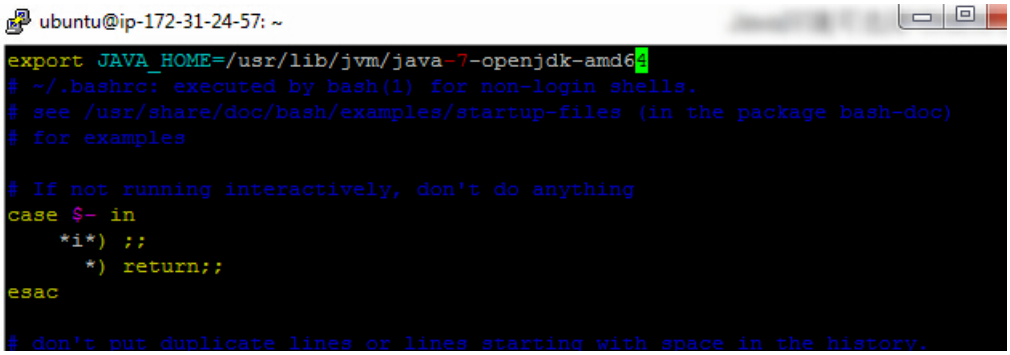


Figure 3.12: Set JAVA Environment Variables

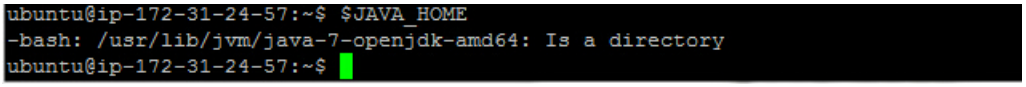


Figure 3.13: Successfully Set JAVA Environment Variables

You can type:

$JAVA_HOME

to test, if the JAVA installation path shows, it means the path is set correctly.

3.1.5 Install Hadoop

The next step is to install Hadoop application on this node (only installation, not configuration). The current stable version of Hadoop can be found at web: http://mirrors.cnnic.cn/apache/hadoop/common/stable/. 2.7.1 is the latest stable version when writing this paper. Please download the file with name hadoop-x.x.x-tar.gz, this is the one that finished compilation. The one with name hadoop-x.x.x-src.tar.gz is the source code which requires you to compile first before using it.(In all the following steps, if you are not using 2.7.1 version, please change the command to your version accordingly)

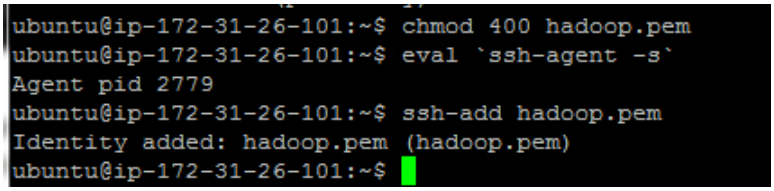


Figure 3.14: Set up Password-less Access

When we get the hadoop-x.x.x.tar.gz file, we need to frist unzip it into /usr/local directory with command:

sudo tar -zxvf ./hadoop-2.7.1.tar.gz -C /usr/local

Also change its owner and group information from root to current user with commands:

cd /usr/local

chown -R ubuntu:ubuntu ./hadoop-2.7.1

In Hadoop directory, type:

./bin/hadoop.

If the help instructions show on the screen, it confirms the Hadoop application is installed correctly. For other 2 nodes in this cluster, we need to perform the same installation procedure for other two nodes. After this, we now have 3 nodes with Hadoop installed. The next step is to configure the cluster.

3.1.6 Set up Configuration of Hadoop

Now we already have 3 nodes with all necessary applications installed, but this is not enough. To have a working cluster, we still need to edit some configuration files to make this cluster correctly running.

3.1.6.1 Set up Password-less Access

On Hadoop cluster, the master node needs to remotely access all slave nodes, but the access between EC2 nodes needs verification (the key pair).

```
<configuration>
<property>
    <name>fs.defaultFS</name>
    <value>hdfs://172.31.26.101:9000</value>
</property>
<property>
    <name>hadoop.tmp.dir</name>
    <value>file:/usr/local/hadoop/tmp</value>
</property>
</configuration>
```

Figure 3.15: Core-site.xml

To enable this password-less access, we need first load the key pair into our nodes. To do this, we need to use a file transfer tool. In this case, I used the tool called WinScp. For information about how to use WinScp, please refer this web: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/putty.html#Transfer_WinSCP

After the .pem file is loaded into our EC2 nodes, we need to first change its permission, otherwise we will get an error saying the permission is too open sometimes later. To do so, type the following command:

chmod 400 xxx(path to your pem file)

Then we need to launch the ssh-agent by typing the commands:

eval 'ssh-agent -s'

ssh-add xxx(path to your pem file)

Now we can easily ssh into any other node in the cluster without being asked to provide the key pair. (ssh-agent and ssh-add commands do not work for lifetime, we need to do it every time we log in)

3.1.6.2 Set up Configuration Files

For a basic Hadoop cluster, we need to, at least, change 5 configuration files. The 5 files are slaves, core-site.xml, hdfs-site.xml, mapred-site.xml, yarn-site.xml. I will talk about how to configure them one by one.

```
<configuration>
<property>
    <name>dfs.namenode.secondary.http-address</name>
    <value>172.31.26.101:50090</value>
</property>
<property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/usr/local/hadoop/tmp/dfs/name</value>
</property>
<property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/usr/local/hadoop/tmp/dfs/data</value>
</property>
<property>
    <name>dfs.replication</name>
    <value>2</value>
</property>
</configuration>
```

Figure 3.16: Hdfs-site.xml

```
<configuration>
<property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
</property>
</configuration>
```

Figure 3.17: Mapred-site.xml

1)slaves: Delete the **localhost** value in original file, and add all the slave nodes IP in it. In our case, two IPs need to be added: 172.31.26.102 and 172.31.26.103.

2)core-site.xml: two properties need to be set. **fs.defaultFS**: need to be set to master nodes IP with port number 9000. **Hadoop.tmp.dir**: need to be set to the temporary directory under Hadoop. My file is set to as 3.15:

3)hdfs-site.xml: 3 properties need to be set. **dfs.namenode.secondary.http-address**: need to be set to Masters IP and port 50090. **dfs.namenode.name.dir and dfs.datanode.data.dir**: corresponding temp file. **dfs.replication**: the number of slave nodes. My file is set to as 3.16:

```
<configuration>
<property>
    <name>yarn.resourcemanager.hostname</name>
    <value>172.31.26.101</value>
</property>
<property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
</property>
</configuration>
```

Figure 3.18: Yarn-site.xml

```
ubuntu@ip-172-31-26-101:/usr/local/hadoop-2.7.1$ jps
3990 SecondaryNameNode
4135 ResourceManager
3763 NameNode
4392 Jps
```

Figure 3.19: Jobs on Master Node

4)mapred-site.xml: one property needs to be set. **mapreduce.framework.name**: need to be set to the resource manage framework: YARN. My file is set to as **??**:

5)yarn-site.xml: 2 properties need to be set **yarn.resourcemanager.hostname**: need to be set to masters IP address. **yarn.nodemanager.aux-services**: need to be set to value mapreduce_shuffle My file is set to as 3.18:

Now we can start to launch our hadoop cluster by the following 3 commands:

bin/hdfs namenode -format

sbin/start-dfs.sh

sbin/start-yarn.sh

3.1.7 Test the Cluster

To test whether the Hadoop cluster is launched successfully, we can type the command:

Jps

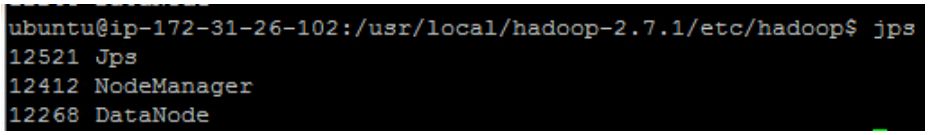


Figure 3.20: Jobs on Data Node

to see all the currently running processes. There should be 3 jobs on master node: **ResourceManger**, **SecondaryNameNode** and **NameNode**.

And 2 jobs on slave node: **DataNode** and **NodeManger**

We can also run the report on Master node by executing the command:

bin/hdfs dfsadmin -report

to check whether slave nodes are running correctly. In my case, two nodes are running as 3.21:

3.2 Auto-deploy Hadoop Cluster on AWS

In last section, we discussed on how to deploy a Hadoop cluster in a regular way. In this paper, our goal is to make this procedure easier so that a client can easily launch a Hadoop cluster with any number of nodes(under AWS limitation) on AWS by just 1 click. In my application, I decided to choose the **AWS SDK FOR JAVA**[16] as the basic framework. It is a JAVA library that richly contains APIs to help you perform operations on EC2 nodes. I used the Eclipse toolkit version for this SDK. Assume your JAVA environment has already been set and Eclipse IDE is installed. In Eclipse, click the Help → Install New Software. Put the link: `http://aws.amazon.com/eclipse` in the work with bar and press enter. Eclipse will show all available packages on that link, pick the **AWS Toolkit for Eclipse** and click next to install it. For more information about this SDK, please refer to web: `http://docs.aws.amazon.com/AWSToolkitEclipse/latest/GettingStartedGuide/welcome.html`

Later, we shall go through and analyze all steps in last section to see whether these steps can be done by programming. If so, how can this be done? In the last

```
ubuntu@ip-172-31-26-101:/usr/local/hadoop-2.7.1$ bin/hdfs dfsadmin -report
Configured Capacity: 16619864064 (15.48 GB)
Present Capacity: 12023865344 (11.20 GB)
DFS Remaining: 12023816192 (11.20 GB)
DFS Used: 49152 (48 KB)
DFS Used%: 0.00%
Under replicated blocks: 0
Blocks with corrupt replicas: 0
Missing blocks: 0
Missing blocks (with replication factor 1): 0

-------------------------------------------------
Live datanodes (2):

Name: 172.31.26.102:50010 (ip-172-31-26-102.us-west-2.compute.internal)
Hostname: ip-172-31-26-102.us-west-2.compute.internal
Decommission Status : Normal
Configured Capacity: 8309932032 (7.74 GB)
DFS Used: 24576 (24 KB)
Non DFS Used: 2343542784 (2.18 GB)
DFS Remaining: 5966364672 (5.56 GB)
DFS Used%: 0.00%
DFS Remaining%: 71.80%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Thu Oct 08 22:07:31 UTC 2015


Name: 172.31.26.103:50010 (ip-172-31-26-103.us-west-2.compute.internal)
Hostname: ip-172-31-26-103.us-west-2.compute.internal
Decommission Status : Normal
Configured Capacity: 8309932032 (7.74 GB)
DFS Used: 24576 (24 KB)
Non DFS Used: 2252455936 (2.10 GB)
DFS Remaining: 6057451520 (5.64 GB)
DFS Used%: 0.00%
DFS Remaining%: 72.89%
Configured Cache Capacity: 0 (0 B)
Cache Used: 0 (0 B)
Cache Remaining: 0 (0 B)
Cache Used%: 100.00%
Cache Remaining%: 0.00%
Xceivers: 1
Last contact: Thu Oct 08 22:07:31 UTC 2015
```

Figure 3.21: Data Node Report

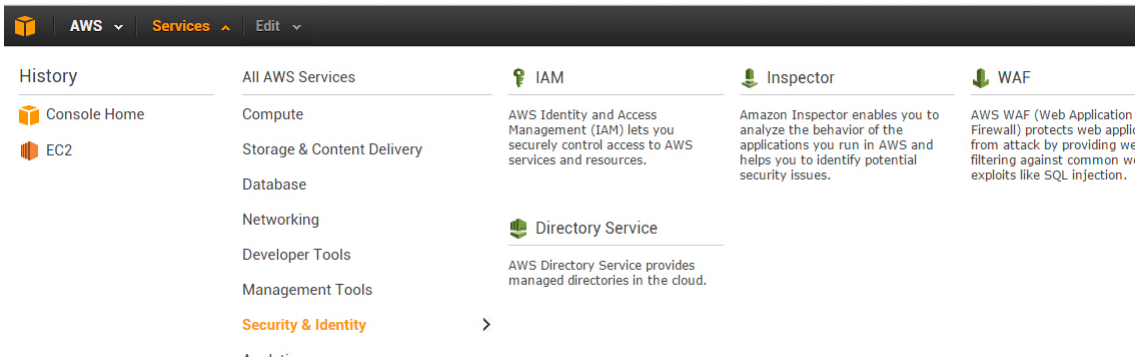


Figure 3.22: IAM of AWS

section, set up Hadoop cluster is divided into 6 major steps:

1.Create an Amazon account

2.Launch AWS instances

3.Log into instances

4.Set up Hadoop cluster

5.Set up configuration of Hadoop

6.Test the cluster

Step 6 can be skipped, now we shalll look into step 1-5 as follows:

3.2.1 Create an Amazon Account

Amazon account can be easily created. In order to allow the AWS sdk get the authorization to access our account, we need to set the AWS Access Credentials. First go to IAM console which is located under Security & Identity service. Click the user button in next page. Then create a user, and the credential will be downloaded into your computer. Save it into the path: C:\Users\your-user-name\.aws\credentials

3.2.2 Launch AWS Instances

This step mainly consists of 3 parts: build a security group, create the key pair and set the appropriate parameters to launch AWS instances.

Create security and key pair in the preparation work. They can be set easily, and not necessarily done by programming.

Launch instances with appropriate parameters: in normal case, we launch a AWS instance with a default AMI and install the JAVA and Hadoop on it. In JAVA and Hadoop installation (do not include the Hadoop configuration) we are doing the same procedure. So if we are going to build Hadoop cluster more than once in future, or we are going to create a cluster with huge amount of nodes, it's better to create a custom AMI which has the JAVA and Hadoop pre-installed. So next time, we can directly launch instances. I will talk about how to set up a custom AMI later. Launch instance can be done by the API: **RunInstancesRequest** provided by AWS sdk.

3.2.3 Log into Instances

After the AWS instances are created, in last section, we use PuTTy to log into them to do some operations. In our auto-deploy application, we need to find a way to communicate with these AWS instances. More specific, we need to find a JAVA library which can support the SSH communication between our client and the remote nodes. After comparing several candidates and making a careful consideration, I picked the Jsch[17] as the SSH library we are going to use. Jsch is written by JAVA and with the goal to implement SSH2. Basically, with Jsch, you can remotely log into a server, transfer files between remote servers and local machine and X11 forwarding, etc. For more information about Jsch, please refer to its website: http://www.jcraft.com/jsch/ It provides a very good example to follow: http://www.jcraft.com/jsch/examples/Exec.java.html

3.2.4 Set up Hadoop Cluster

Set up Hadoop cluster (not including setting up configuration files) consists of two parts: intall JAVA and install Hadoop application. Basically, these need to be done on each newly launched node. However, there is another very easy way to do this, the custom AMI. Amazon Machine Image(AMI) can be considered as a template,

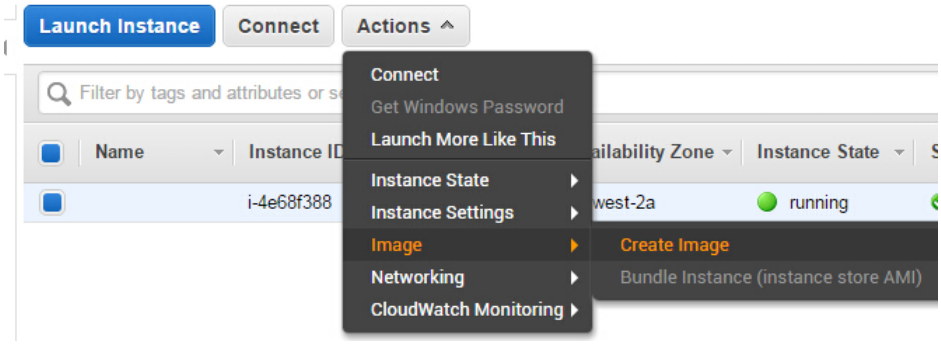


Figure 3.23: Create Custom AMI-Step 1

which contains all the information needed to launch a new node, including the type of operating system, the pre-install applications, etc. The one we used in manually deploy part is a default Linux AMI provided by Amazon. However, we can create our own AMI. With such AMI, when a node is created, it will be same as the custom AMI with all the required applications, such as JAVA and Hadoop, are installed. To create a custom AMI, we first set up everything on a sample node. (install and configure all applications we want to be on the new node) and in the AWS EC2 panel, choose the sample node and pick: Actions → Image → Create Image.

Select a name for it, and create Image. When a custom image is created, it can be used to launch a new instance. So in our case, we need to create a custom AMI with the JAVA environment and Hadoop application. Then the only thing we need to is to configure the Hadoop file if we plan to set up a Hadoop cluster in future. Furthermore, some standard configuration files, (the files content is always the same and does not depend on its role (master or slave) or other dynamic elements like the IPs). We will talk about this in the next section.

3.2.5 Set up Configuration of Hadoop

As we discussed in the previous section, some parts of the configuration can be pre-configured and build into the custom AMI. Lets take a look at each part of the

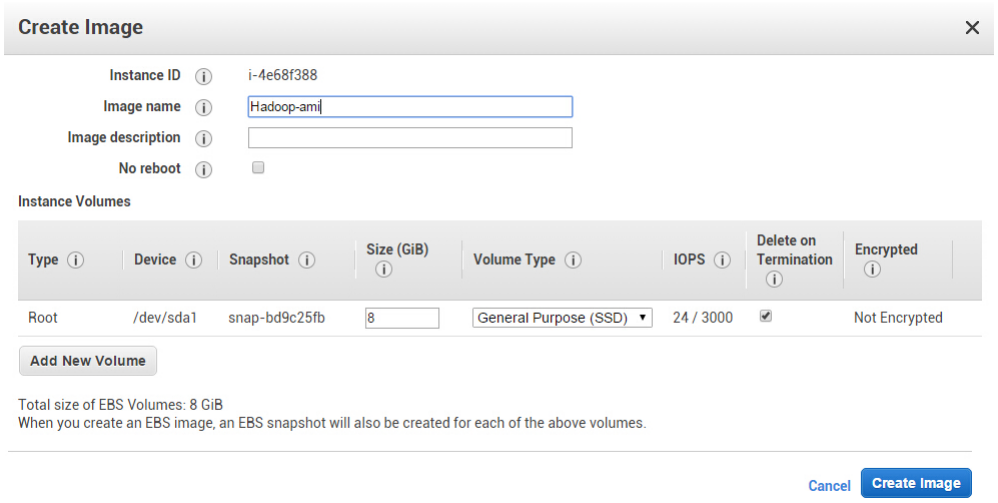


Figure 3.24: Create Custom AMI-Step 2

configuration file in details:

1) slaves: we need to firstly delete the localhost value and fill in all the IPs of the slave nodes. The IPs are dynamically allocated so they cannot be built into the custom AMI. So we leave blank in this file.

2) core-site.xml: two properties need to be set. **fs.defaultFS**: need to be set to master nodes IP and port number 9000. The masters IP can not be pre-determined,

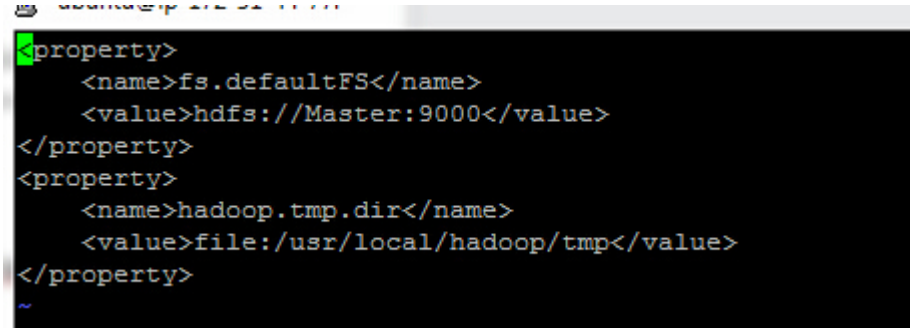


Figure 3.25: Core-site.xml with Undetermined Master Node IP

```
<property>
    <name>dfs.namenode.secondary.http-address</name>
    <value>Master:50090</value>
</property>
<property>
    <name>dfs.namenode.name.dir</name>
    <value>file:/usr/local/hadoop/tmp/dfs/name</value>
</property>
<property>
    <name>dfs.datanode.data.dir</name>
    <value>file:/usr/local/hadoop/tmp/dfs/data</value>
</property>
<property>
    <name>dfs.replication</name>
    <value>replication</value>
</property>
```

Figure 3.26: Hdfs-site.xml with Undetermined Master Node IP

```
<property>
    <name>mapreduce.framework.name</name>
    <value>yarn</value>
</property>
```

Figure 3.27: Mapred-site.xml, No Change Needed

so it can not be built into custom AMI. We fill a keyword Master in the position of IP. **Hadoop.tmp.dir**: need to be set to the temporary directory under Hadoop. This directory is fixed, so this value can be built into custom AMI. The file is set to as 3.25:

3) hdfs-site.xml: 3 properties need to be set. **dfs.namenode.secondary.http-address**: need to be set to master's IP and port 50090. Same as core-site.xml, the masters IP can not be pre-determined, so it can not be built into custom AMI. We fill a keyword Master in the position of IP. **dfs.namenode.name.dir and dfs.datanode.data.dir**: corresponding temp file. The directory is fixed so this can be built into the custom AMI. **dfs.replication**: the number of slave nodes. This number is decided by the client and is different from clusters. So it can not be built into custom AMI. We fill the keyword replication in the position of number. So the file is set to as 3.26:

```
<property>
    <name>yarn.resourcemanager.hostname</name>
    <value>Master</value>
</property>
<property>
    <name>yarn.nodemanager.aux-services</name>
    <value>mapreduce_shuffle</value>
</property>
```

Figure 3.28: Yarn-site.xml with Undetermined Master Node IP

4) mapred-site.xml: one property needs to be set. **mapreduce.framework.name**: need to be set to the resource manage framework: yarn. This is fixed, so it can be built into custom AMI. So the file is set to as **??**:

5) yarn-site.xml: 2 properties need to be set **yarn.resourcemanager.hostname**: need to be set to masters IP address. Same as above, the masters IP can not be pre-determined, so it can not be built into custom AMI. We fill a keyword Master in the position of IP. **yarn.nodemanager.aux-services**: need to be set to value mapreduce_shuffle. This is fixed, so it can be built into custom AMI. So the file is set to as 3.28:

With the above analysis, we now have a custom AMI ready with required application installed (Hadoop), required environment set (JAVA) and all configuration files ready and their values can be pre-determined. So right now, to launch a Hadoop cluster is easier, we only need to set up the configuration files which needs the run-time information (IPs).

Chapter 4

OVERVIEW OF THE PROGRAM

The main contribution of this thesis is a application that automatically deploy a Hadoop cluster on AWS instances. In previous sections, we perfromed a detailed analysis on how to make such application. In this chaper, let's take a detailed look at the code level. I will show some codes for the key functions in the program and give explainations for them if necessary. The source code for the program is uploaded to Github.com, it can be found at: https://github.com/chudel2015/my-thesis

4.1 Prerequisite

Before we start to discuss the program, we need to make sure the following preparations are finished:

1: Custom AMI: as we discussed in section 3.2.0.4, the custom AMI has the Hadoop and JAVA installed, JAVA environment ready and some configurations pre-set if these configuration values are fixed.

2: AWS access Credential: as we discussed in section 3.2.0.1

3: Security group: as we discussed in section 3.2.0.2

4: Pair Key: as we discussed in section 3.2.0.2

4.2 Program Structures

In this section, I will show codes and give some explainations follow the program sequence.

4.2.1 Set up AWS Credentials

The first thing I did in my program is to set up the AWS credentials, because no AWS operations are allowed without it. There are two ways to set the credentials:

```
credentials = new ProfileCredentialsProvider("default").getCredentials();
AmazonEC2 ec2 = new AmazonEC2Client(credentials);
```

Figure 4.1: Set Up Credential-1

```
credentials = new BasicAWSCredentials("AKIAJ24E42EOSPY3UBLQ", "fw7ync7CwSt4FsK4CT4IxLl/lPqTMm0wv8fF/hnE");
AmazonEC2 ec2 = new AmazonEC2Client(credentials);
```

Figure 4.2: Set Up Credential-2

1: You can place the credentials under the directory C:\Users\your-user-name\aws\credentials and provide the value default to API **ProfileCredentialsProvider**.

2: You can directly provide the credentials values to the API **BasicAWSCredentials**.

After finishing setting the credentials, we need to provide the credentials to the main API **AmazonEC2Client**.

4.2.2 Gather User Data

After setting up the credentials, we now can access the AWS account. We need user provide some important information, such as:

What is custom AMI‘s name?

How many nodes to be launched this time?

What is pair key‘s name?

```
int cluster_size =Integer.parseInt(JOptionPane.showInputDialog("Enter number of machines you want to set"));
String ami_name = JOptionPane.showInputDialog("Enter your ami name");
String key_name = JOptionPane.showInputDialog("Enter your key name");
String s_group_name = JOptionPane.showInputDialog("Enter your security group name");
```

Figure 4.3: JOptionPane

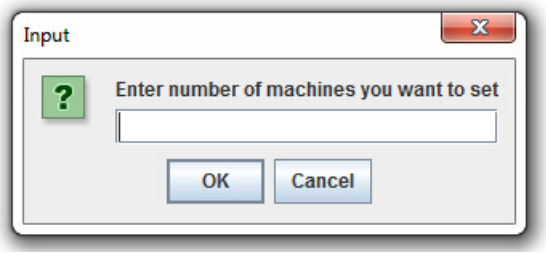


Figure 4.4: Dialog Box

```
RunInstancesRequest runInstancesRequest = new RunInstancesRequest();
runInstancesRequest.withImageId(ami_name)
                   .withInstanceType("t2.micro")
                   .withMinCount(cluster_size)
                   .withMaxCount(cluster_size)
                   .withKeyName(key_name)
                   .withSecurityGroups(s_group_name);
```

Figure 4.5: Fill User Data

What is security group's name?

In our program, we used **JOptionPane** from the **javax.swing** libary, several dialog boxes will jump out to ask user the above questions. Then collected information will be filled into the API **RunInstancesRequest**.

4.2.3 Launch Instance and Collect Necessary Information

To launch AWS instances, simply execute the **runInstances** function from **AmazonEC2Client** with the **RunInstancesRequest** provided.

```
RunInstancesResult runInstancesResult = ec2.runInstances(runInstancesRequest);
```

Figure 4.6: Launch Instance

```
Thread.sleep(10000);

Vector<String> instanceId = new Vector<String>();
for(Instance ins : runInstancesResult.getReservation().getInstances())
  instanceId.add(ins.getInstanceId());

DescribeInstancesRequest request =  new DescribeInstancesRequest();
request.setInstanceIds(instanceId);
DescribeInstancesResult result = ec2.describeInstances(request);
List<Reservation> reservations = result.getReservations();

List<Instance> instances_list = new Vector<Instance>();
for(int i=0; i<reservations.size(); i++)
  instances_list.addAll(reservations.get(i).getInstances());
```

Figure 4.7: Collect Instance Information

4.2.4 Collect Instance Information

After we launch the AWS instances, we need to collect some instance information, such as the instance IPs for future use. To do so, we need first suspend the program for several seconds because it takes time to launch the requested instances, the instance information are not available immediately. After few seconds, we are able to gather the instanceIDs from API **RunInstancesRequest**. With the instanceIDs, we are able to collect the reservation through the APIs **DescribeInstancesRequest** and **DescribeInstancesResult**. The reservation contains all the instances‘ information.

4.2.5 Log into Instance

After the AWS instances are launched, the next step is to log into them and make appropriate configurations. In order to do that we need to log into the instances first through a library called **Jsch** as we discussed in section 3.2.0.3. To log into instances, we need provide the Jsch our private pair key and add it as identity. After the private key is provided, we need to create a SSH session, and the connection to the instance can be established if some correct information are filled in(user name, IP and port number)

```
JSch jsch=new JSch();
JFileChooser chooser = new JFileChooser();
chooser.setDialogTitle("Choose your privatekey");
chooser.setFileHidingEnabled(false);
int returnVal = chooser.showOpenDialog(null);
if(returnVal == JFileChooser.APPROVE_OPTION) {
  System.out.println("You chose "+
   chooser.getSelectedFile().getAbsolutePath()+".");
  jsch.addIdentity(chooser.getSelectedFile().getAbsolutePath()
 );
}
```

Figure 4.8: Select Private Key

```
Session session;
UserInfo ui=new MyUserInfo();
for(int i=0; i<instances_list.size(); i++)
{
  if(instances_list.get(i).getPublicIpAddress()==null)
      System.out.println("Error, public ip is null\n");

  System.out.println("Connect to:"+instances_list.get(i).getPublicIpAddress()+"\n");
  session = jsch.getSession("ubuntu", instances_list.get(i).getPublicIpAddress(), 22);
  session.setUserInfo(ui);
  session.connect();
```

Figure 4.9: Log into Instance

4.2.6 Hadoop Configuration

Now, the SSH connection is established, the final job is to edit the Hadoop configuration files with corresponding information. There are 4 files need to be modified: salve, core-site.xml, hdfs-site.xml and yarn-site.xml.

One simple way is to echo all required configuration content into an empty file. But this is not recommended, because this build-in information is not standardized. The configuration procedure we discussed in previous sections is just a sample configuration. There are various ways to configure a Hadoop cluster. As we discussed in section 3.2.0.5, the better way is to leave a keyword in the configuration file in custom AMI, such as Master stands for IP of master node. And we replace such keyword with the real time information. To replace the key words, there are three methods:

1: We can transfer the configuation file back to local machine, then we can replace the keyword on local machine. When the replacement is finished, we transfer the configuration file back to AWS instance to replace the old file. Here is a very good example showing how to use Jsch to transfer files: http://www.jcraft.com/jsch/examples/ScpTo.java.html.

2: We can build a string replacement program and build it into the custom AMI. When the instanced is launched, we can replace the keyword by executing this program.

3: We can replace the keyword in a file by the **SED** command, for how to use sed command in Linux, here is a good tutorial: http://www.brunolinux.com/02-The_Terminal/Find_and%20Replace_with_Sed.html

In our program, we pick the third method. After all files are properly configured, we can disconnect the SSH session and the Hadoop cluster is ready to be used now.

```
//slaves file
for(int j=0; j<instances_list.size(); j++)
{
  if(j!=0)
    exec("echo "+instances_list.get(j).getPrivateIpAddress()+
    "\n >> /usr/local/hadoop/etc/hadoop/slaves",session);
}
//core-site file
String command = "sed -i 's#Master#"+instances_list.get(0).getPrivateIpAddress()+
"#g' /usr/local/hadoop/etc/hadoop/core-site.xml";
exec(command,session);

//hdfs-size file
command = "sed -i 's#Master#"+instances_list.get(0).getPrivateIpAddress()+
"#g' /usr/local/hadoop/etc/hadoop/core-site.xml";
exec(command,session);

command = "sed -i 's#replication#"+Integer.toString(cluster_size-1)+
"#g' /usr/local/hadoop/etc/hadoop/core-site.xml";
exec(command,session);

//yarn-size file
command = "sed -i 's#Master#"+instances_list.get(0).getPrivateIpAddress()+
"#g' /usr/local/hadoop/etc/hadoop/core-site.xml";
exec(command,session);
```

Figure 4.10: Edit Configuration Files

Chapter 5

CONCLUSION

This paper has given an in-depth insight to automatically fast deploy a Hadoop cluster with any number of AWS EC2 nodes. A detailed introduction was given on the history of Hadoop, including its first generation MRV1 and next generation MRV2. Having explained the importance of Hadoop, our goal is to provide a solution to easily and quickly set up a Hadoop cluster with a cloud computing service. In our methodology,, we first went through a tutorial on Hadoop installation. However, for cloud computing service, the node‘s information(such as IPs) will be different when we request a new set of nodes, so it is time-consuming to finish the installation, especially for large size of cluster. There was an analysis of the procedure and proposed a new method to achieve the above goal. Some constant settings, such as JAVA installation, can be included into the AWS custom AMI. And the communication between local machine and remote nodes is implemented under the help of the Jsch library. With the help of the proposed tool, people are able to focus on more important research work by freeing themselves from the time-consuming installation and configuration.

BIBLIOGRAPHY

[1] Douglas Laney. 3d data management: Controlling data volume, velocity and variety. *Gartner*, February 2001.

[2] Bringing big data to the enterprise. https://www-01.ibm.com/software/data/bigdata/what-is-big-data.html.

[3] Martin Hilbert. Big data for development: A review of promises and challenges. *Development Policy Review*, January 2013.

[4] Ecl programming language. https://en.wikipedia.org/wiki/ECL_programming_language.

[5] Hadoop poweredby. http://wiki.apache.org/hadoop/PoweredBy.

[6] A.-M. Popescu V. Ubarhande and H. Gonzalez-Velez. Novel data-distribution technique for hadoop in heterogeneous cloud environments. *Complex, Intelligent, and Software Intensive Systems (CISIS), 2015 Ninth International Conference on*, July 2015.

[7] S. Bailey S. Narayan and A. Daga. Hadoop acceleration in an openflow-based cluster. *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion*, November 2012.

[8] Peng Ning Yu Xianqing and M.A. Vouk. Enhancing security of hadoop in a public cloud. *Information and Communication Systems (ICICS), 2015 6th International Conference on*, April 2015.

[9] ChangJun Jiang Dazhao Cheng, Jia Rao and Xiaobo Zhou. Resource and deadline-aware job scheduling in dynamic hadoop clusters. *Parallel and Distributed Processing Symposium (IPDPS), 2015 IEEE International*, May 2015.

[10] Hardesty E. Hai Jiang Jie Zhu, Juanjuan Li and Kuan-Ching Li. Gpu-in-hadoop: Enabling mapreduce across distributed heterogeneous platforms. *Computer and Information Science (ICIS), 2014 IEEE/ACIS 13th International Conference on*, June 2014.

[11] A. Rasooli and D.G. Down. A hybrid scheduling approach for scalable heterogeneous hadoop systems. *High Performance Computing, Networking, Storage and Analysis (SCC), 2012 SC Companion*, November 2012.

[12] T. Ragunathan T.L.S.R. Krishna and S.K. Battula. Performance evaluation of read and write operations in hadoop distributed file system. *Parallel Architectures, Algorithms and Programming (PAAP), 2014 Sixth International Symposium on*, July 2014.

[13] K. Daudjee L. Northam, R. Smits and J. Istead. Ray tracing in the cloud using mapreduce. *High Performance Computing and Simulation (HPCS), 2013 International Conference on*, July 2013.

[14] John Wiley and Sons. Data science and big data analytics: Discovering, analyzing, visualizing and presenting data. *ISBN 9781118876220*, December 2014.

[15] Matei Zaharia Michael J. Franklin Scott Shenker Reynold S. Xin, Josh Rosen and Ion Stoica. Shark: Sql and rich analytics at scale. *SIGMOD*, June 2013.

[16] Aws sdk for java. https://aws.amazon.com/sdk-for-java/.

[17] Jcraft. http://www.jcraft.com/jsch/.