UiO : Department of Informatics
University of Oslo

Cost Efficient Batch Processing in Amazon Cloud

Cost minimization on batch processing with deadline using spot instances in Elastic MapReduce Cluster

Kabin Tamrakar

Master's Thesis Spring 2016

Cost Efficient Batch Processing in Amazon Cloud

Kabin Tamrakar

May 23, 2016

Abstract

Cloud Computing provides computing and storage resources at economical price with flexibility, mobility and availability. These resources range from small capacity to very high capacity computes. The cloud providers also offer spare compute instances at significantly low price.

Amazon Cloud Service provider has a popular bidding scheme on their spare computes called spot instances which can be requested with bid price. The spot instances are vulnerable to termination at any time if spot market price exceeds the bid price. Amazon also rents on-demand instances which are persistent with fixed price. Spot instance price may drop up to 90% compared to price of on-demand instance. In this project, spot instances are resorted in task instances' group of Amazon EMR cluster to process batch jobs with deadline. Amazon EMR makes it convenient to process big data by the aid of managed Hadoop framework. The processed intermediate results in the task nodes of the cluster are lost if the spot instances gets terminated that can cause processing delay.

The cost efficiency can be realized by exploiting non-real time nature of batch computing for Big Data. Two algorithms are devised for achieving cost efficient processing in Hadoop MapReduce. Both algorithms process data in divisions such that abrupt termination of spot instances affects that division only. Based on progress at some interval and checkpoints, task group's capacity is resized to complete processing in time. Progress is completion of number of divisions of work. The first algorithm begins with spot instances in estimated quantity. To complete processing of all data in time, on-demand instances are employed after threshold time. The second algorithm starts by using higher number of spot instances than required to complete the work within deadline. It has higher probability to utilize only spot instances because of faster work progress. On-demand instances are deployed only in case of slow progress. The experiments show that both algorithms minimize the cost of processing. The second algorithm further minimizes the cost in most cases.

Acknowledgements

Cloud computing is one of the hotcakes of the modern information technology. Working on this thesis, I learnt a lot about recent cloud services and system automation. It has opened doors for me to research more in this field during my professional life.

First of all, I would like to express my sincere gratitude to my supervisors Hårek Haugerud and Anis Yazidi for the continuous guidance, remarks, engagement through the learning process and close monitoring of my master thesis. I would like to take this opportunity to thank them and express how immensely important were their lecture, weekly meeting, suggestion for completion of my master's thesis. The positive spirit from my supervisors has always encouraged me to thrive more, work hard and realize my potential.

I would also like to mention my professor Kyrre Begnum's contribution for enhancing my research skills, knowledge about recent technologies in our field through his amazing lectures and notes. Information Technology is the most dynamic industry. He has deeply influenced me to be updated with the modern technology.

Big Thanks to my fellow classmates for the all the help, support, important discussions, motivation. They have inspired me to work hard, helped me when I was stuck in some technical problems. I want to remember how were they available to help during my thesis, anytime via social media. My sincere gratitude goes to University of Oslo (UiO) and Oslo and Akerhus Universiy College (HiOA) for providing quality education and infrastructures.

I would like to thank my wife Mamta Maharjan for all her love, support and interest. She has always inspired me with insightful discussions and constructive suggestions. Thanks for keeping up with my busy schedule and taking care of my health. Last but not the least, my unlimited gratitude goes to my mom, dad and sisters. I cannot thank them enough for all the care, love, moral support, encouragement.

- Kabin Tamrakar

Contents

List of Figures

List of Tables

Acronyms

The following acronyms are used in this report:

- AMI - Amazon Machine Image
- AWS - Amazon Web Services
- AWS CLI - Amazon Web Services Command Line Interface
- CW - Cloudwatch
- DAAB - Deadline Aware Auto Bidding Scaling Algorithm
- DPB - Deadline Aware Progress Adaptive Burst Bidding Algorithm
- EC2 - Elastic Compute Cloud
- EMR - Elastic MapReduce Cluster
- HDFS - Hadoop Distributed File System
- IaaS - Infrastructure as a service
- JAR - Java Archive
- MR - Map Reduce
- OD - On Demand
- PaaS - Platform as a Service
- S3 - Simple Storage Service
- SaaS - Software as a Service
- SLA - Service Level Agreements
- VM - Virtual Machine
- VPC - Virtual Private Cloud
- YARN - Yet Another Resource Negotiator

Chapter 1

Introduction

Cloud computing has become indispensable for both big and small enterprises to perform numerous IT operations today. For dynamic workloads, it is often economical to rent cloud servers than building and configuring dedicated infrastructure [27]. Major advantages it offers are reduced costs, automation, flexibility, mobility and consumerization however it has security and confidentiality as major concerns. In addition to bringing beneficial aspects to cloud users in terms of costs, flexibility and availability, it brings unique challenges to cloud service providers themselves. Cloud users may demand significant resources during peak hours and peak computation. Service provider guarantees that sufficient cloud resources are available to meet the service level agreements (SLA) with cloud users. This means the cloud providers have to arrange significantly large resource pool to serve the users' demand anytime but during off-peak hours there is a significant waste of cloud infrastructure [43]. To cope with the unutilized resouces, cloud service providers offer different pricing options so as to facilitate a wide variety of applications depending on computing requirements [32, 45].

The common cloud pricing schemes for virtual machine compute instances are namely reserved, on-demand and spot instances [32]. Reserved instances provide users with a one-time payment and get instances over a long period of 1-3 years and also receive hourly discounted pricing based on usage [32]. While on-demand instances are offered as an hourly instances without any long-term commitment. For spot instances, cloud spot markets ask users to bid for spare resources without any guarantee on termination. The cloud provider can revoke the spot resources once the market price exceeds the bid price [32, 45]. With higher risks and uncertainty of being revoked anytime, the spot resources are by as much as 10X cheaper than the equivalent on-demand resources which cannot be revoked by cloud providers for paid hours [45].

Any of the above compute instances can be used for a large

variety of workload use-cases like always-on Internet based services, batch processing, transaction processing, analytics, high performance computing, database computing, etc which can be performed in the cloud. The cloud spot resources can be implemented to perform these use-cases with large monetary benefits. However spot resources may not be always available. There can be price fluctuation on the basis of supply-demand paradigm which results into immediate termination by cloud provider as soon as bidding price become lesser than spot market price. The system should deploy an effective bidding algorithm along with fault-tolerant mechanisms and switching to on-demand resources to maintain its availability and reliability. He et al. [27] implemented two types of bidding algorithm - reactive and proactive along with some migration mechanisms. They concluded that proactive algorithm achieve goal of acceptable unavailability of service because this algorithm aids in graceful and planned migration.

This project will provide light into the batch processing with deadline using amazon spot instances. Amazon Elastic MapReduce (EMR) cluster will be implemented for processing the batch jobs. Amazon EMR's Hadoop Cluster is composed of Master Instance Group and Slave Instance Groups - Core and Task. Master nodes and core slave nodes should be running on consistent instances as master nodes constitutes the central part of cluster and core slave nodes also runs YARN ResourceManager service for application resources management and HDFS NameNode service. Task nodes group can be resized anytime so that for shorter deadline needs, the capacity can be increased and vice versa. Even core nodes group can be resized but shrinking them may cause the risk of losing data as they store data in HDFS.

A stable Amazon EMR cluster with a core group having at least one consistent instance and other as spot instances and task group with all spot instances can be devised. Therefore on the basis of availability of spot instances within the bidding price, the jobs may be completed earlier or late. However, it is complicated for batch workloads with deadline. The system should be devised in such a way that for a given deadline, it starts the processing with Task Instance Group of spot instances. Depending on the status of completed percentage of work, task capacity needs to be varied. It may even require on-demand instances to complete processing tasks within deadline. Effective algorithms will be devised for the provision of the required amount of instances. For the deadline based batch workloads, an additional work on estimating approximate processing time will be required.

1.1 Problem Statement

The following problem statement will be implemented to materialize the vision of this thesis:

How can a cost efficient MapReduce cluster be designed and developed in Amazon by resorting to spot instances for batch processing with deadline?

For answering this problem statement, several assumptions have been made based on the background theory and technologies. The detail answers to the problem statement is provided in **Approach** chapter by breaking down the stated problem statement into multiple sub-questions and each sub-questions are answered.

1.2 Structure of the Report

The report is divided into following chapters:

- **Introduction**: The problem domain and scopes are explained in this chapter.
- **Background**: This chapter discusses background theoretical and technological bases and related literature review.
- **Approach**: This chapter reflects the methodologies and general approaches and technologies being followed to answer problem statement along with challenges.
- **Results I** - Design: The two algorithms are explained with detail algorithms and expected results.
- **Results II** - Implementation and Experiments: Number of experiments were carried out after implementing the algorithms from previous chapter as tool. This chapter explains system implementation details, experiments and their results.
- **Results III - Analysis**: The experiment results from previous chapters are evaluated and analyzed in this chapter. Analyses on time and costs are made with data and graphs.
- **Results III - Discussion**: This chapter discusses implementation steps and challenges. It also light on future work.
- **Results III - Conclusion**: This chapter gives summary of the project.
- **Appendix**: Developed python scripts and AWS CLI commands are presented.

- **Acronyms**: The acronyms used throughout this paper are mentioned in this chapter for convenience to the reader. Acronym chapter is placed before Table of Contents.

Chapter 2

Background

2.1 Cloud Computing

The phrase "Cloud Computing" is becoming the trending buzzwords in information technology for past few years and appearing more than 173 million[1] times in internet websites. There are lot of debates on when and who coined the term "Cloud Computing" for the first time which went back to the dates to the 1960s in context of network based computing, but it is believed that first use of this term in its modern context took place in 2006 during an industry conference [25] by then Google CEO [37]. The following year its scope started to widen with different sales efforts in cloud introduced by companies Amazon, Microsoft and IBM.

Cloud Computing is an internet-based computing which provide on-demand application, platform and infrastructure as resources. With pay-as-use principle, huge investment on installation and maintenance of IT infrastructure has dropped dramatically. The growth is significantly leaping because of the availability of high-capacity networks, cheaper computing machines and storage device along with advent of virtualization, service-oriented architecture, autonomic computing and concepts of utility computing [42, 29]. Businesses and end users enjoy elasticity of cloud as they can scale up or down conveniently on the basis of their requirement and demands.

Cloud Computing is often described in wrong way as **Virtualization**. But there are distinctions between Cloud Computing and Virtualization. Virtualization is a platform that enables to run multiple operating systems on a single physical system with accessiblity to use the underlying hardware on sharing [55]. The goal of virtualzation is the workload management by adapting it to more scalable, efficient and economical via transformation in traditional computing [31]. Virtualization is the founda-

[1]Google Search Engine statistics to search "Cloud Computing" as on 03/03/2016

tion of cloud computing because it is the enabling technology to deliver cloud resources.

2.1.1 Different models of Cloud

It comes in three different models namely private, public and hybrid clouds. A private cloud maintains services and infrastructure on a private network with higher security level and control. On the other hand public cloud is a cloud where services and infrastructure are rendered over internet. Data is stored in the service providers' data center and they are responsible of operating and maintaining all the components. Security and confidentiality is prime concern in public clouds. Finally, hybrid clouds integrate both private and public clouds from multiple providers. The benefit of using hybrid cloud minimises the trade-offs and also maximize performance with the flexibility it offers.

2.1.2 Different service categories

Different cloud services are categorized as Infrastructure as a Service (IaaS), Platform as a Service (PaaS) and Software as a Service (SaaS). IaaS provides hardware, software, servers, storage and other infrastructure. IaaS services are self-service models and allow users to access, monitor and manage remote datacenter infrastructures conveniently and users can pay for infrastructures based on consumption like utilitiy billing. Amazon Web Services (AWS), Microsoft Azure and Google's Compute Engine are some examples of IaaS.

Platform as a Service (PaaS) provides computing platforms. Users can use remote operating systems, environment for compiling and executing programming languages, web services and database services, etc. Windows Azure, Google App Engine and Heroku are some examples of PaaS. The benefits PaaS offers is reduced complexity, effective application development, built-in infrastructure along with convenient maintenance and enhancement of application. It can also be attributed by the fact that lot of resources and time spent on environment setup before the actual software development is no more concern.

Software as a Service (SaaS) provides cloud application services. Users can access the web interfaces managed by third-party vendor and use the services. The beauty of SaaS applications is that they can be run directly from a browser without extra installations and downloads or just need to have plugins. So enterprises can efficiently and smoothly provide their maintenance and support. Google Apps, Cisco WebEx and Citrix GoToMeeting are some examples of SaaS.

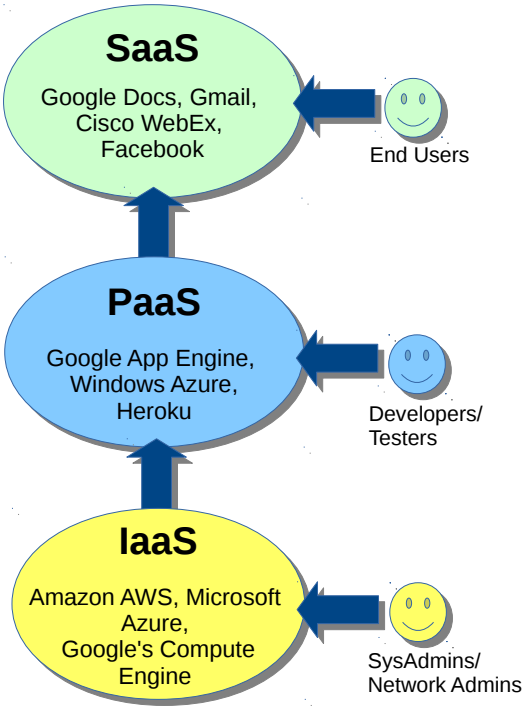

Figure 2.1: Different Cloud service models.

2.1.3 Cloud Service Providers

There are number of cloud service providers today providing different cloud options. Out of different cloud services, Cloud Infrastructure as a Service is the main concern in this thesis. The most popular and leading vendors of IaaS cloud services are Amazon with Amazon Web Services, Microsoft with Azure Infrastructure Services, Google with Compute Engine, CenturyLink with their managed sevices, VMWare with vCloud Air and Rackspace with their managed services. Choosing the right vendor depends on customer's use cases as different vendors are specialized in different use cases.

Amazon Web Services has the biggest share in the IaaS market, with the highest IaaS compute capacity in use than the other IaaS vendors [46]. According to Gartner[2] report published in October 2015, Amazon Web Services receives a 4.81/5.0 for application development Use Case, 4.81/5.0 for Batch Computing Use Case, 4.84/5.0 for Cloud-Native Applications Use Case and 4.53/5.0 for General Business Application Use Case. This was done by comparing 15 cloud IaaS services against eight

[2]Gartner - the world's leading IT research and advisory company [54]

critical capabilities in four different use cases [39]. It is recommended by Gartner to use Amazon Web Services for all use cases except for strictly compliant applications.

Azure is recommended for general business applications and development environments for Microsoft-centric organizations and also suited for cloud-native applications and batch computing [46]. While Google is recommended for cloud-native applications and batch computing. Gartner recommends to use CenturyLink for self-service cloud IaaS with managed services and applications excluding batch computing. vCloud Air is to be used for development environments, business applications and to supplement VMware-virtualized environments.

2.1.4 Amazon Web Services

Amazon Web Services (AWS) is collection of cloud computing web services to offer cloud-computing platform of Amazon.com. The AWS clouds are available for operation in 33 availability zones at 12 geographic regions around the world. It is in pipeline that 10 more availability zones and 5 more regions are coming online by the next year. Amazon Web Services is continually improving their services with following practices [24]:

- High Availability through Multiple Availability Zones
- Improving Continuity with Replication between Regions
- Meeting Compliance and Data Residency Requirements
- Geographic Expansion

The list of different products AWS offers in categories Compute, Networking, Storage and content delivery, Database, Deployment, Management, Application Services and Analytics. Elastic Compute Cloud (EC2), Elastic MapReduce (EMR), Route 53, Virtual Private Cloud (VPC), Elastic Load Balance (ELB), Elastic Block Storage (EBS), DynamoDB, OpsWorks are some of the products Amazon offers in above different categories.

2.1.4.1 Amazon Elastic Compute Cloud

Amazon Elastic Compute Cloud (EC2) is a web service provided by Amazon which enables customer to use re-sizable compute capacity in the cloud. It forms a major part of Amazon's cloud-computing platform. EC2's web interface allows user to deploy, configure and run virtual machine via Amazon Machine Image [7] and it is called a compute instance. It also make

it easier for user to have complete control over their computing resources which are run in Amazon's proven environment [22]. The time needed to get and boot new instances is within minutes, that enhances scaling capacity, both up and down as required. With the pay as you go (PAYG) model, EC2 charges only for capacity that is actually being used.

EC2 provides number of built-in security features. The created instances are located in Amazon's Virtual Private Cloud (VPC) [9] which is logically isolated network. VPC takes care of security features like Network Access Control Lists, AWS Identity and Access Management (IAM) Users and Permissions and Security Groups. Amazon Elastic Block Storage (EBS) can be used to provide persistent block storage to EC2 instances. EC2 also provides developers with the tools to build and run fault tolerant applications.

The benefits of using EC2 are following [22]:

- Elastic Web-Scale Computing
- Complete Control of instances
- Flexible Cloud Hosting Services
- Convenient conjunction with other Amazon Web Services
- Reliability with Service Level Agreement of 99.95% availability
- Secure by conjunction with Amazon VPC
- Inexpensive compute instances with Pay as you go (PAYG) model
- Easy kickoff with preconfigured software on Amazon Machine Images (AMI)

2.1.4.2 Amazon EC2 Purchasing Options

Amazon EC2 deploys Xen Virtualization. Each EC2 virtual machine is called EC2 instance, which is virtual private server (VPS) [51]. There are three purchasing options for Amazon EC2 instances namely On-Demand, Reserved and Spot Instances. There is also one other option in which dedicated EC2 instance capacity on physical servers is provided termed as Dedicated Hosts [36]. Functionally all these instances perform identically.

For **on-demand instances**, there is a payment for a compute capacity on hourly basis with no long-term commitments or upfront costs. Compute capacity can be increased or decreased any time and payments have to be made only for the used instances at specified hourly rate. Amazon EC2 manages on-demand capacity to be available most often, otherwise it is possible that specific on-demand instance types in specific

availability zones may not be available for short time. A **dedicated host** is fully dedicated physical EC2 server instance. The dedicted hosts enable in saving costs on server-bound software licenses, and compliance and regulatory requirements. It can be purchased as hourly on-demand.

For **reserved instances**, amazon EC2 assure that it will always be available for the Operating System and Availability Zone which is purchased. It can be reserved for 1 or 3 years [3]. There is significant discount in price of up to 75% in reserved instances as compared to on-demand instance. It is recommended for applications with steady state or predictable usage as it can leverage significant profits than using on-demand instances. It is possible to move the reserved instance purchased to other availability zone within the same region, to alter its network platform, or modify instance type to another type of same family in linux/unix without additional cost.

For **spot instances**, there is a process of bidding to purchase spare EC2 compute capacity without upfront costs and hourly rates are usually lower than the on-demand pricing schemes. Maximum hourly price has to be specified by the bidder to run particular instance type. Amazon EC2 sets a spot price for each type of instances in each Availability Zone for a given period which is dynamically changing on the basis of supply and demand for instances. The spot price can be as low as just 10% of on-demand price [4]. AWS assesses availability of number of spot instances in each pool, and at the same time monitors the bids from different bidders. On this basis, AWS provisions the available spot instances to the bidders with maximum bidding prices. In case the spot price becomes greater than the bidders' maximum bid price, Amazon EC2 will terminate the instances.

List of following different types of use cases can be performed by using spot instances [48]:

- Batch Processing - Generic background processing
- Hadoop - Hadoop processing type jobs
- Scientific Computing - Scientific trials, simulations and analysis
- Video and Image Processing/Rendering - Video transformation
- Testing - Software/websites testing
- Web/Data Crawling
- Financial - Trading analysis
- High Performance Computing - Utilize HPC to do parallel jobs
- Cheap Compute - Backend servers for web games

2.1.4.3 Amazon Simple Storage Service

Amazon Simple Storage Service also referred as Amazon S3 provides with highly scalable cloud service [8]. It provides a simple web interface that can be used to store and retrieve data from the internet. Tools like Amazon Web Services Command Line Interface (AWS CLI) can be used for handling this storage. The S3 service can be implemented as single service or integrated together with other Amazon Web Services. It can be used as a persistent data store for keeping input data and results of the cluster. The pricing ranges from $0.0275/GB to $0.0300 per GB per month depending on the amount of data in Oregon (us-west-2) region.

2.1.4.4 Boto - Amazon Web Services SDK

Boto is AWS SDK for python programming language. It provides platform to integrate Python scripts and library with AWS services so that Python developers can develop software to make use of AWS services like Amazon EC2 and Amazon S3. There is latest release of next version of Boto as Boto3 which is stable and recommended for use. Compatibility of Boto3 along with Boto in the same project makes it convenient to work in existing project made in boto.

2.1.5 Google Preemptible Instances

Google provides a low priced instance option from their spare compute resource. These instances also offers same machine types as other instances Google rent out. Unlike spot instances are acquired by the method of bidding, it has always a fixed low price which is 70% cheaper compared to other regular instances. Hence they can be used for expanding the computing capacity at low price. However similar to spot instances they can be terminated any time by the cloud provider if demand for other instances increases.

2.1.6 Bidding spot instances

There is provision to buy spare EC2 compute capacity with heavily reduced price in Amazon Web Services. Market price is set for every available spot instances of different sizes like "m1.medium", "m3.xlarge", "c4.xlarge", etc [21]. The bidding procedure is discussed in "spot instances" portion of subsection 2.1.4.2. Technically, with desired bid price spot instance is requested to Amazon Web Services (AWS) [44]. On request submission, if there are bad parameters it will terminate. Otherwise a unique spot request ID is generated and its status changes to *pending-evaluation*. It can go to

closed state if conditions are *capacity-not-available*, *price-too-low*, *az-group-constraint*, etc. If there is availability and the bid price is above market price, the request is fulfilled and instances got launched. The running instances may be terminated by user requests itself, or by price, capacity, etc. It may also run persistently if any of these constraints does not arise.

The figure 2.2 shows market price over time and its effect on spot instances. At time 0, bid was successful because the bid price was higher than market price and at x time, the market price just exceeds bid price, the instance got terminated. If the spot instance is terminated by Amazon itself, only partial time x is charged.

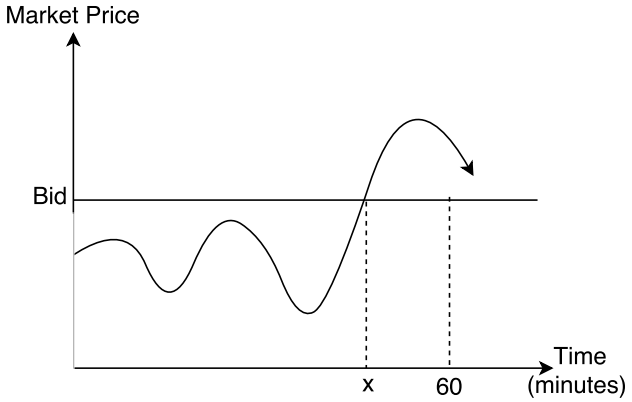

Figure 2.2: Successful Spot Bidding and Termination

2.2 Clustering

A computer cluster is a group of computers and hardware connected together as parallel or distributed computing system in order to unleash a single virtual and powerful hardware platform. The benefits it provides are much faster processing speed, increase in storage capacity, improvement in data integrity, better reliability and wider availability of resources [53]. Basically, it enables high availability, load balancing and parallel processing. The motivation for cluster computing is the necessity and is facilitated by the advent of commodity processors with high performance, high-bandwidth and low-latency networks and sophisticated software tools and development infrastructures [15]. Flexibility is another characteristic as unlike mainframe computers, computer clusters can be adapted easily to enhance or diminish the existing specs or add or remove the component/s itself to the system.

On the basis of deadlines and price constraints, demand of resources may get increased or decreased in this project. When there is larger deadline, it is possible to wait for low priced spot instances if they are not available outright otherwise to meet deadline, even high priced

instances will have to be provisioned to get processing done in time. With static partitioning there is dedicated set of resources for particular task and it cannot efficiently address the situation as it may not address deadline constraint where the processing need to be addressed by additional virtual machines either cheaper spot instances or on-demand instances. The cluster is aimed for some peak traffic setup, then the resources granted for peak timing would be over-provisioned for the condition when there is very low traffic. Clustering gives the added layer of abstraction and thus with the use of software it is possible to partition hardware dynamically. Without degradation of performance caused by the underlying hardware partitioning, services running on top of a cluster can efficiently scale and dynamically move within the cluster [56].

2.2.1 Hadoop

Hadoop is an open source Java based programming framework that enables processing of huge sets of data in distributed computation. The basis of Hadoop is to scale from single servers to thousands of local machines providing computation and storage locally. Distributed computation refers to computer clusters built on the top of commodity hardware. The hardware failures are automatically handled by the Hadoop to provide fault tolerant system. It promises to deliver a highly available service on top of a cluster of those individual servers which individually are failure-prone. Hadoop implements MapReduce programming model.

The principle behind **MapReduce** model is to first divide data into many small fragments of works and each of them would be performed on any node in the cluster as map. And finally in second step is to organize and reduce the results generated into an aggregate answer to a query. One master called Job-tracker and many workers called Task-trackers constitute the MapReduce architecture.

The large or complex sets of data can be referred to as **Big Data** for which traditional data processing applications are not sufficient. Big Data can be structured, semi-structured and unstructured data which can be analyzed and mined for useful results and decisions [52]. In this project, huge data of system logs is regarded as Big Data to be processed with Hadoop. The focus is on deadline and scaling of servers rather than on logic of data analysis.

2.2.2 Amazon Elastic MapReduce

Amazon Elastic MapReduce (EMR) functions as a web service which is used for analyzing and processing a huge set of data. The computational work is distibuted across a cluster of AWS virtual servers and the cluster is

managed by Apache Hadoop [49], an open source framework. EMR makes it convenient to process big data by the aid of managed Hadoop framework otherwise it is difficult to configure and tune Hadoop clusters. The Hadoop project, that includes MapReduce, YARN (Yet Another Resorce Negotiator) and HDFS (Hadoop Distributed File System) are installed and configured programatically across the cluster nodes by EMR. This service enable to process data-intensive tasks quickly. It is also posible to develop and run sophisticated applications by the aid of scheduling, workflow and monitoring facilities.

As discussed in last paragraph, Amazon EMR implements **Apache Hadoop** as its distributed engine to process data. Hadoop is an open source library and framework developed in java for distributed processing of large sets of data across large clusters of commodity hardware.

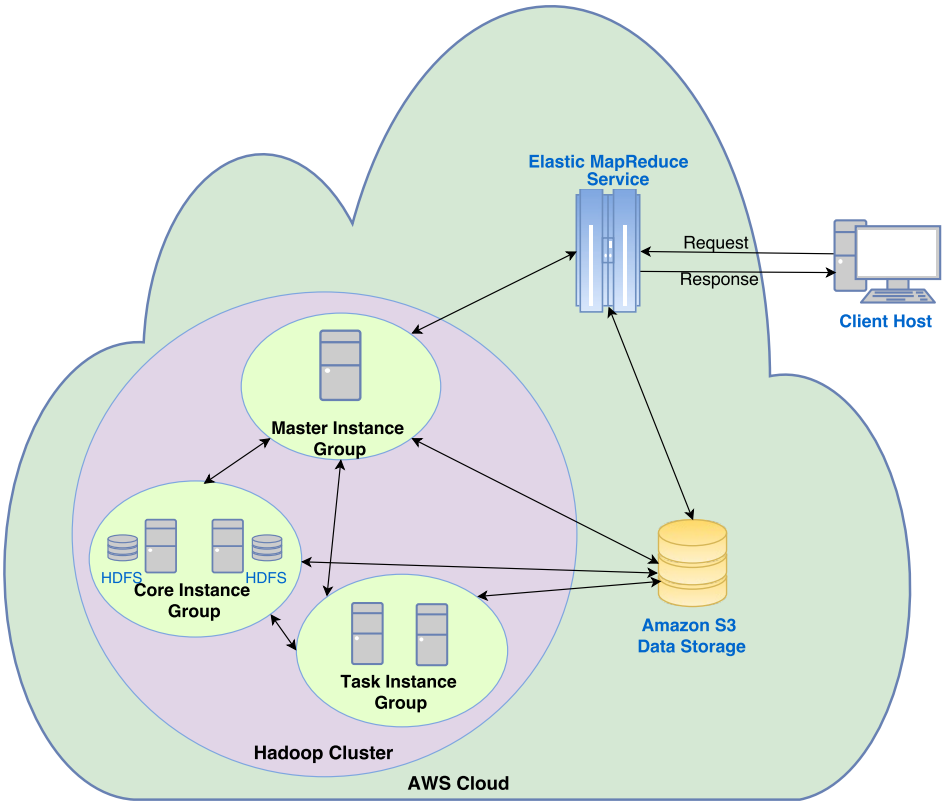

Figure 2.3: Typical Amazon Elastic MapReduce Cluster.

In Amazon EMR cluster, there is conceptualization of instance groups composed of groups of EC2 instances and they perform according to the roles defined by the distributed applications installed on the cluster [28]. The groups belong to master and slave groups. The instance groups are of three types viz., master itself as Master Instance Group, and slave groups divided into two groups namely Core Instance Group and Task Instance Group.

The master Instance Group is used for managing the cluster and also to run master components of the distributed applications installed on the cluster. It starts the YARN (Yet Another Resource Negotiator) [10] Resource Manager service and the HDFS (Hadoop Distributed File System) NameNode service. Another job is to assess and track status of jobs being processed in the cluster and keep track of health status of the instance groups. For hadoop mapreduce jobs, elements on core and task nodes perform the data processing, and generated output is transferred to Amazon S3 or HDFS. Finally status metaback is sent to the master node.

The core instance group consists of the core nodes of a cluster and each node runs the tasks. By running the DataNode daemon it stores data as part of HDFS. The master node manages the core nodes. When there are no YARN jobs or applications running, the "shrink" operation will attempt to diminish the instances in the group after getting a resize request for that instance group.

Task instance group consists of the task nodes of a cluster and they are optional groups. Task groups can be added when cluster is started or can be added into a running cluster. They do not store data in HDFS. They can be used to handle peak loads by increasing its capacity.

2.2.2.1 Amazon EMR Metrics

Amazon EMR sends the number of metrics to Amazon CloudWatch [5]. It pulls metrics from a cluster but no metrics are reported in case the cluster becomes unreachable. The table provides Hadoop 1 metrics on Cluster status.

There are more metrics on the basis of *Nodes Status*, *IO Status* and *HBase* in Hadoop 1 and in Hadoop 2 Amazon Machine Images (AMIs). The Hadoop 1 metrics are tabulated in table **2.2**.

Hadoop 1 metrics on Cluster Status	
Metric	**Description**
IsIdle	A cluster is not performing. It is still in on state and charging rent **Use:** *Cluster Performance Monitoring* **Units:** *Boolean*
JobsRunning	The number of running jobs in cluster **Use:** *Cluster Health Monitoring* **Units:** *Count*
JobsFailed	The number of failed jobs in the cluster **Use:** *Cluster Health Monitoring* **Units:** *Count*

Table 2.1: EMR Metrics on status of Cluster

Hadoop 1 metrics	
Metric	**Description**
MapTasksRunning	The number of map tasks running for each job **Use:** *Cluster Progress Monitoring* **Units:** *Count*
MapTasksRemaining	The number of map tasks remaining for each job **Use:** *Cluster Progress Monitoring* **Units:** *Count*
MapSlotsOpen	The remaining map task capacity. It is difference of maximum number of allocated map tasks for a given cluster and current number of map tasks running. **Use:** *Cluster Performance Analysis* **Units:** *Count*
ReduceTasksRunning	The number of reduce tasks that are running for each job **Use:** *Cluster Progress Monitoring* **Units:** *Count*
ReduceTasksRemaining	The number of reduce tasks that are remaining for each job **Use:** *Cluster Progress Monitoring* **Units:** *Count*
CoreNodesRunning	The count of core nodes which are running **Use:** *Cluster Health Monitoring* **Units:** *Count*
TaskNodesRunning	The number of working task nodes **Use:** *Cluster Health Monitoring* **Units:** *Count*
S3BytesRead	The size in bytes of data read from Amazon Simple Storage Service **Use:** *Cluster Performance Analysis and Health Monitoring* **Units:** *Bytes*

Table 2.2: EMR Metrics for Hadoop 1

Hadoop 2 AMI has different metrics than Hadoop 1. They can be referred from table **2.3**.There are more metrics than presented in the tables. The EMR metrics tell about Cluster progress, Cluster performance and Cluster Health. So tracking the cluster becomes convenient. Number of decisions can be made on the basis of these metrics which are sent to **CloudWatch**. The integration of CloudWatch with AWS CLI and Python APK Boto eases developer to control and manage the cluster in the way it is needed.

Hadoop 2 metrics	
Metric	**Description**
ContainerAllocated	Resource containers allocated in the cluster by ResourceManager **Use:** *Cluster Progress Monitoring* **Units:** *Count*
ContainerPending	The containers which are in queue to be allocated **Use:** *Cluster Progress Monitoring* **Units:** *Count*
AppsCompleted	The completed tasks submitted to YARN **Use:** *Cluster Progress Monitoring* **Units:** *Count*
CoreNodesRunning	The count of core nodes which are running **Use:** *Cluster Health Monitoring* **Units:** *Count*
MRTotalNodes	The available number of map reduce nodes **Use:** *Cluster Progress Monitoring* **Units:** *Count*
MRActiveNodes	The number of MapReduce nodes running **Use:** *Cluster Progress Monitoring* **Units:** *Count*
MRDecommissionedNodes	The MapReduce nodes that has gone to DECOMMISSIONED state. **Use:** *Cluster Monitoring* **Units:** *Count*
HDFSBytesRead	The size in bytes of data read from HDFS **Use:** *Cluster Performance Analysis and Health Monitoring* **Units:** *Bytes*

Table 2.3: EMR Metrics for Hadoop 2

2.2.3 CloudWatch

To track the progress and health of running EMR cluster, CloudWatch is introduced Amazon Web Services which collect and track metrics, collect and monitor log files and also set alarms for auto alerting against unwanted changes in the resources. As it gives system-wide visibility on different aspects like resource utilization, health and performance, it can be useful for smooth operations of running applications. Amazon EMR records metrics and sends them to the Amazon EMR console and also to the CloudWatch console. Metrics updation takes place every five minutes which is not configurable and the metrics are pushed to CloudWatch for every Amazon EMR cluster.

2.3 Relevant Theory/Literature Review

There have been significant number of researches on efficient utilization of spot market to cut down the operation costs in many applications. Spot instances have fluctuating price based on supply demand concept and irregular availability based on available spare cloud compute capacity. The major setback could be its termination as discussed earlier. Thus to make a stable system out of such instance is challenging, and many factors need to be considered and studied. Following are some research which are relevant and related.

2.3.1 Cutting the Cost of Hosting Online Services Using Cloud Spot Markets

This research written by He et al. [27], focuses on minimizing the cost of running always-on internet-based services by the use of spot markets. It is to be noted that at least four nines i.e. 99.99% of availability is widely accepted standard to tag the Internet-based service as always-on.

The cloud scheduler developed by them has less time to migrate the service to on-demand resulting in more disruption to the service when they use reactive bidding algorithm. In reactive bidding algorithm, migration takes place after spot server is revoked. The authors therefore designed their proposed proactive bidding algorithm which senses the varying spot market beforehand for gracefully shutting down of spot instances and migration to on-demand instances.

For both reactive and proactive algorithm , they uses three migration steps namely Forced Migration, Planned Migration and Reverse Migration on the conditional basis of current spot price, bidding price and on-demand price. For the migration from spot to on-demand and

vice versa, OS mechanisms such as Nested vitrualization, Live migration, Bounded Memory checkpointing and Lazy VM restore were used. Both proactive and reactive bidding causes in significant cost reduction as cost is just 17% to 33% of the cost if all on-demand instances were being used. And unavailability of the service using the proactive algorithm is smaller by a factor of 2.5 to 18 than using the reactive algorithm. So later they use proactive algorithm to bid with multi-region strategy and the cost reduced to 12% to 17% of the baseline cost using only on-demand virtual machines.

2.3.2 Automated cloud bursting on a hybrid cloud platform

This is Master's thesis paper written by Xue [56]. In this paper, the author emphasizes in building highly available cluster in hybrid cloud. For handling additional compute for spikes during peak hours, an automated cloud bursting solution in public cloud is developed which uses amazon spot instances to leverage from its pricing model. Basically the paper deals with setting up a hybrid cloud using Apache Mesos to make a unified platform for private alto cloud and amazon public cloud focusing on mazimizing availability. For acquiring spot instances to serve additional compute, an automated script is written which implements a simple bidding algorithm. There is no performance analysis though it is mentioned in the report as future work.

2.3.3 WOHA: Deadline-Aware Map-Reduce Workflow Scheduling Framework over Hadoop Clusters

Li et al. [30] developed WOHA, a framework that efficiently schedule deadline-aware workflows in MapReduce. For simplification of submitting workflows, workflow scheduler like Apache Oozie has emerged so that Hadoop has to handle only resource allocation and Oozie workflow topology but it still lengthen the workflow spans causing deadline misses. They have presented a solution as WOHA which makes client nodes to generate scheduling plans locally and sent to master node which will use it for scheduling plans. They propose a scheduling algorithm which assign priorities among workflows and for its evaluation they cover three job prioritizing algorithms - namely Highest Level First (HLF), Longest Path First (LPF) amd Maximum Parallelism First (MPF).

2.3.4 Towards a MapReduce Application Performance Model

This research carried out some benchmark performance testing on MapReduce applications [26]. Even though MapReduce has emerged easier to use for huge data analytics and characterized robust as fault tolerance be-

ing automatically handled by the runtime system, the performance can be questioned with the presence of individual machine failures. These failures can cause significant delays in execution of jobs as they have to be rescheduled into new nodes. They did a benchmark test with *mrbench* by starting with a single, fixed-work trivial map or reduce task. Then in next iteration with two and continued for some higher numbers. It came up with the overhead on application performance. Another benchmarking tool *mrbench-waves* which can cause each compute node to run only on one non-trivial map or reduce task at a time. Using *mrbench* benchmark, the map task over-head was found to be nearly 0.111 seconds and reduce task overhead to be 0.105 seconds in an experimental setup of 34 compute nodes. Since typical MapReduce clusters consists of hundreds of compute nodes, these overheads can grow very significantly.

2.3.5 How to Bid the Cloud

In this research, the authors suggested that calculating effective bid price depending on jobs' interruption constraints can decrease cost of processing significantly with spot instances and less job interruption [57]. For determining the bid price, 2 months' statistics of spot market price was utilized. The authors employed one-time requests and persistent request as bidding strategies for master and slave nodes in MapReduce jobs. One time request was made for a single spot instance with high bid price. As interruption is allowed in slave node, persistent request was made for each slave node. The bid price for slave nodes was calculated on the basis of master node since master nodes need to run more time than slave nodes. From the results, they derived that their bidding strategies could adapt to cluster based MapReduce jobs with significant low price. The bidding strategies developed in this research assumed that single user's bid price would not affect the market price. However in case of multiple users' bidding the market price would be affected. So it was suggested to use the mathematical model developed to extract the effect of multiple bidding on the market price.

Part I

The project

Chapter 3

Approach

This chapter will outline and explain the methodologies, processes and general approaches being followed in order to answer the defined problem statement: ***How can a cost efficient MapReduce cluster be designed and developed in Amazon by resorting to spot instances for batch processing with deadline?***

3.1 Objectives

The problem statement can be addressed as a whole by breaking down the question into several sub-questions and answering each of them. The sub-questions can be following as specified in the introduction chapter:

- How can a computer cluster be built in Amazon?
- How can huge batch jobs be processed in the cluster?
- What metrics can be used to monitor progress and estimate total time to process data?
- How can decision be made for varying processing capacity by increasing and decreasing processing nodes to meet deadline?
- Is it possible to use spot instances only to meet deadline?

3.2 Design Stage

The system will be designed to achieve goal of the problem statement. Two algorithms will be formulated and developed as automation tools. For the experiments these tools will be deployed and on the basis of measured data

analysis will be made. The algorithms will answer last two and important problem statements ***How can decision be made for varying processing capacity by increasing and decreasing processing nodes to meet deadline?*** and ***Is it possible to use only spot instances to meet deadline?*** The expected components and tools are discussed in section **3.3** while first three problem statements are answered individually at project phases in subsections **3.4.2**, **3.4.3** and **3.4.4** in section **3.4**.

Initially, an experiment will be done with a static cluster unaware of any automation. For this a static cluster will be designed with desired number of task nodes with on-demand instances only in order to carry out processing job within deadline. Since on-demand node provides a consistent cluster throughout its lifetime, this will also become benchmark for analyzing the performance, benefits and shortcomings of the tools to be implemented with the algorithms.

The first algorithm will be aware of deadline and will scale the cluster at ten regular intervals. It will begin with estimated number of spot instances (refer to **3.4.4**) in the first interval. Afterwards from the second interval, required new task nodes capacity will be calculated on the basis of progress and elapsed time. The cluster will be rescaled with the same bidding price as before. This will repeat in every interval till the second last interval. Before the final interval, task capacity will be recalculated and only on-demand instances will be used to guarantee the completion of processing in time.

The second algorithm will also be aware of deadline with burst bidding. Bidding more spot instances by some factor α will be done from beginning in order to complete processing earlier with spot instances only. However in the worst case, it will require on-demand instances as well. Besides checkpoint will be used at 50% of elapsed time and 80% of elapsed time. Hence depending upon the progress at these checkpoints different strategies will be implemented. The first algorithm will have a static bidding price calculated at the beginning while in the second algorithm, there will be dynamic bid price calculated on the basis of latest price summed with x percentage of difference between on-demand price and latest price. At the checkpoints, progress is calculated as "Behind", "On-track" and "Ahead". So for slow progress like "Behind" aggressive bidding would be used i.e. higher value of x for spot instances along with on-demand instances to catch up with deadline. Otherwise, bidding would be done with smaller value of x. Bidding price B would be given by following formula with L as latest spot market price and D as on-demand price:
$B = L + (D - L) \cdot x$

3.3 Necessary Components and Tools to build the models

The system will be designed and developed with different components and tools and by following the distinct phases in order to process deadline based batch jobs with huge processing needs. The first three questions in problem statement are answered in project phases in subsection 3.4.2, section **3.4**.

3.3.1 The Cluster

Amazon Elastic MapReduce (EMR) is a managed hadoop framework cluster provided by Amazon. The detail background about EMR is in subsection **2.2.2**. The application as Core Hadoop with Hadoop 2.7.2 and EMR release emr-4.6.0 [6] is chosen for the thesis. This is the latest release of Amazon EMR.

3.3.2 MapReduce Data Processing Engine

For data processing, some application is required. A log analysis application will be deployed which takes log messages as a data set for analytics in Amazon EMR. The MapReduce based processing engine will be deployed which is made by developer other than author. A Custom Java Archive (JAR) implemented in online book store named Safari Books Online will be deployed for analyzing generated syslog like logs. Java Mapper, Reducer and Driver classes are compiled into a Custom JAR named log-analyzer which expects syslogs as input and output as number of events per second in a text file. The source codes for the Mapper, Reducer and Driver classes are available here [19]. The data flow from syslog file to mapper to reducer to output will give number of events per second which is depicted in figure **3.1** below.

3.3.3 Computing virtual machines

Amazon Elastic Compute Cloud (EC2) instance is chosen as computing virtual machine as the EMR cluster only supports them. Detail background about Amazon EC2 instances is in subsection **2.1.4.1** Instance type chosen for this thesis is "m1.medium" as it is the instance with lowest possible configurations for Hadoop 2 clusters [17]. Only "m1.medium" instance will be used throughout the experiments for using it as master, core and task instances in EMR cluster.

"m1.medium" instance type has following specifications [2]:

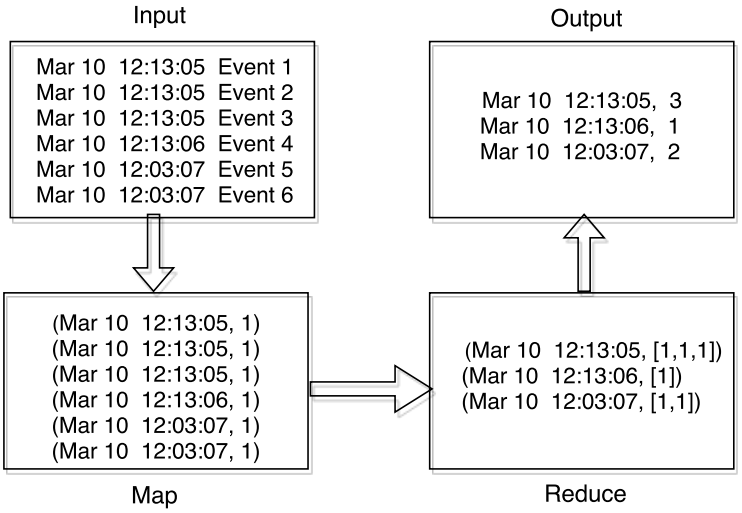

Figure 3.1: Map Reduce Data Flow

Instance Type	**Amazon EC2 m1.medium**
Memory size	3.7 GB
Primary (OS) disk	9.0 GB
Secondary disk	393.0 GB
CPU speed	1.43 GHz
Core concurrency	1.00
Memory bandwidth	5.7 GB/s
Primary disk rate	62 MB/s
Secondary disk rate	114 MB/s
Primary disk seeks	97 ops/s
Secondary disk seeks	75 ops/s

Table 3.1: EC2 m1.medium specifications

"m1.medium" instance with Linux/Unix operating system has on-demand price of **$0.087** per our in Oregon Zone (us-west-2) and its all availability regions. The price at different zones differs. The instance pricing also differs on the basis of operating system chosen.

3.3.4 Persistent Storage

As the system compromises of the amazon EMR cluster and amazon EC2 instances, amazon S3 is preferred to use as persistent storage for storing input data to feed into the EMR cluster and also to store processed data as output from the cluster. It is compatible to be used with either Amazon Web Services or alone. Its detail explanation can be read in subsection **2.1.4.3**.

3.3.5 Monitoring

Monitoring needs to be done for checking cluster and its nodes' health and progress status. Different metrics will be considered for visualizing these statuses as described in subsection **2.2.2.1**. Amazon CloudWatch, referred in subsection **2.2.3** is a convenient tool available to visualize these EMR metrics.

Besides CloudWatch, fetching log files from master node can also be done to realize the cluster progress like checking Maps and Reduce progress, Hadoop Distributed File System (HDFS) read write status on-the-fly unlike Cloudwatch which gets data every five minutes.

3.3.6 Script Development for Automation

For automation, Python Programming is preferred due to author's familiarity with this language and availability of AWS SDK for Python - Boto3 SDK (refer to subsection **2.1.4.4**). For live logs analysis from master node, Paramiko [50] API can be used which is python implementation of Secure Shell 2 (SSH2). Basically Paramiko can be useful to connect to the remote master node via SSH, execute commands in the master node and get the result out from remote node to local machine.

3.3.7 FoxyProxy

Hadoop has its web user interfaces hosted on the master node. For security reasons, it is required to configure proxy settings to view the web interface on the master node [17]. First SSH tunnel has to be created to the master node using dynamic port forwarding. Then FoxyProxy can be configured in different browsers to reach these websites.

3.3.8 R-programming

Graphs and other visualization for analysis of results will be done using R-programming. Microsoft Excel will also be used for data visualization.

3.4 Project Steps

3.4.1 Sample Data Generation

Sample data need to be generated for processing in the EMR cluster. Data can be realized in the form of syslog and will simply be counting the number of events per second. The objectives of this project is to process huge data that require dynamic processing requirements to guarantee timely job completion with less budget. Hence these generated data and analyzed output are not major concern but they contribute as input data for the EMR cluster being engaged in processing.
Each line of standard syslog has following format:

```
Month Day HH:MM:SS "Event with its info"
```

Example:

```
Mar 25 11:28:24 kab-lappy dhclient: Sending on Socket/fallback
Mar 28 13:00:35 kab-lappy ntfs-3g[4969]: Unmounting /dev/sda7(Ent)
Mar 28 13:00:35 kab-lappy udisksd[2525]: Unmounted /dev/sda7 on
    behalf uid 1000
```

In such syslog format, sample log will be generated which would be processed by EMR cluster and the map reduce data flow has already been mentioned in **3.1**.

The sample syslogs can be compressed in gzip, bzip2 or LZO format as these compression types are supported by Hadoop. It will be very useful because uploading the compressed syslogs to Amazon S3 buckets incur low bandwidth and low cost for storing data persistently. Besides, network bandwidth while transferring data from Amazon S3 to HDFS in core nodes will also be significantly low. As Hadoop supports mentioned compression formats, very huge file can be compressed to size of 64 MB (default split size) for unsplittable compression like gzip while even bigger sized compressed files in bzip2 can be used as input files which will be splitted and assigned to individual mapper.

For experiment, huge data is expected to be fed into the processing engine of the EMR cluster. Therefore replication of few of compressed samples generated can be done to make huge number sample logs which will need large computational power and time.

3.4.2 Cluster Setup and Configuration

The first question of problem statement ***How can a computer cluster be built in Amazon?*** will be addressed here. There are many possible ways to setup a cluster in amazon. For an instance, setting up hadoop cluster by using Amazon EC2 instances as Namenodes and Datanodes and manually configuring everything. Other distributed frameworks are also available like Apache Spark [33] and Presto [35] which can be setup and configured manually in Amazon EC2 instances and then process data as per requirement with customized processing engine. Amazon also has another solution which makes it convenient by providing managed Hadoop framework and it is called Amazon EMR cluster which can be referred in subsection **2.2.2**. Even Apache Spark and Presto can be run in Amazon EMR. For this thesis, Amazon EMR will be setup and configured with Core Hadoop 2.7.2 on EMR release 4.6.0. [1].

3.4.3 Map Reduce Engine

To address second question of problem statement ***How can huge batch jobs be processed in the cluster?***, a processing engine has to be realized. Simple log analysis from huge log data will be done. For the experiments sample syslog like logs will be processed with Map Reduce Engine to get the results in the form of number of events per second. The MapReduce engine to be used is mentioned in subsection **3.3.2**.

3.4.4 Estimation Phase

In this phase approximate size of cluster will be calculated based on desired deadline. As "m1.medium" EC2 instance will be implemented for the experiments, so estimation should be done accordingly. This phase also answer the part of third question of problem statement ***What metrics can be used to monitor progress and estimate total time to process data?*** The following steps will be followed to estimate required number of nodes for processing.

- Number of mappers running in parallel in an instance be N_m. Default mappers capacity of "m1.medium" is 2. Similarly other instances like "m1.xlarge" can run 8 mappers in parallel by default. If a cluster consists of 1 master node and 1 core node, the total instance mapper capacity would be N_m since only core and task instances are involved in actual processing unlike master node which involves in coordinating all the nodes.
- What is number of mappers the job requires?
  As discussed in sub-section **3.4.1**, large number of sample log files

x would be created and size can be about 64 MB but should not be exceeded it if the file is unsplittable like compressed file format *gzip*. This means number of splitters and hence mappers would be x.

$$S_m = x$$

- How long would it take to process sample files? It is assumed that it would take t_m time to process m sample files. The thing to be noted is number of sample files to process is equal to number of mappers that can run in parallel in that instance for number of instances calculating formula mentioned in next point.
- Finally with desired deadline t_d, approximate estimated number of nodes [14] would be X_n which is given as below:

$$N_x = \frac{S_m \cdot t_m}{N_m \cdot t_d}$$

This number N_x would be crucial and needs to be maintained during processing. Any number of instances $N > N_x$ running for given period of deadline can make processing happen in time.

3.4.5 Data Processing and Cluster Scaling

Before beginning actual processing, the data should be made available to S3 bucket. Above log generating bash script can be run into an EC2 instance in same region to bucket, compress data and transfer it to S3 bucket. It is also possible to generate sample logs locally and then upload via internet into S3 bucket as in figure **3.2**. Data processing engine has to be ready. The EMR cluster should be up and running. The cluster will be provisioned with at least 1 master node and 2 core nodes. Later on the basis of value of N_x which is required number of task instances to complete the job in time, the cluster will be scaled up and down. The value of N_x will be adjusted itself prior to deadline at defined checkpoints.

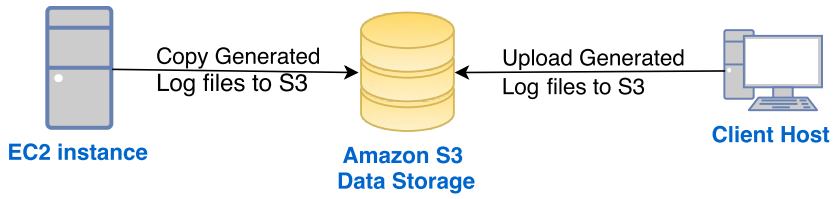

Figure 3.2: Loading data into S3 bucket

The overall system design of the cluster will be like the block diagram depicted in figure **3.3** below.

Utilizing Python boto3 SDK, CloudWatch and log analysis all the activities in the cluster will be monitored and decision for cluster resizing

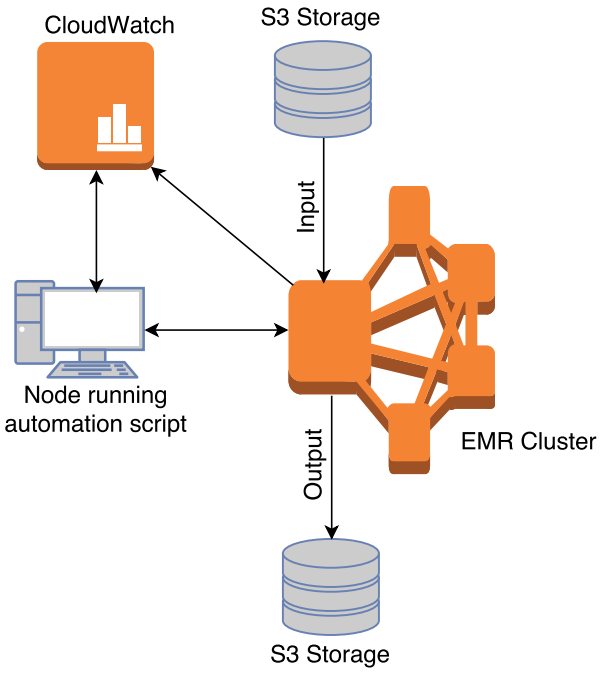

Figure 3.3: System Block Diagram

will be initiated from the node which will be running automation script. For improved performance of the job flows all nodes and S3 bucket for the given cluster should be launched in the same Amazon EC2 Availability zone to incur higher data access rate and to eliminate inter-zone data flow charges.

3.5 Challenges of using Spot instances in Cluster

When a cluster starts processing the data, mappers and reducers are assigned to the slave nodes (core and task instances). The output of the mapper is buffered and later dumped into local disk of each individual mapper nodes as the intermediate results when threshold of buffer is reached. As described earlier, spot instances could terminate any time market price exceeds the bid price or spot pool is empty. When using spot instances as task nodes in the EMR cluster, mappers running on those instances store the intermediate map outputs in those spot nodes itself and the cluster always risk losing those intermediate results.

For instance, there is chance of losing spot instances after processing 70% of data with spot task nodes so these data need to be reprocessed which causes huge loss of both economy and time. Solution to address this huge processing loss can be done by dividing the job into more partial jobs like 10 partial jobs. After completion of one partial job

store the output and start next partial job and so on. This means maximum of 10% processing would be lost as each step contributes 10% of total work. For example if the current step be 4^{th} after finishing 3 partial jobs then 30% work has already been finished. In case spot instances get terminated, only partial percent above 30% of work would be lost and this partial percent of work could be recovered with new spot instances or on-demand instances in time to avoid missing deadline.

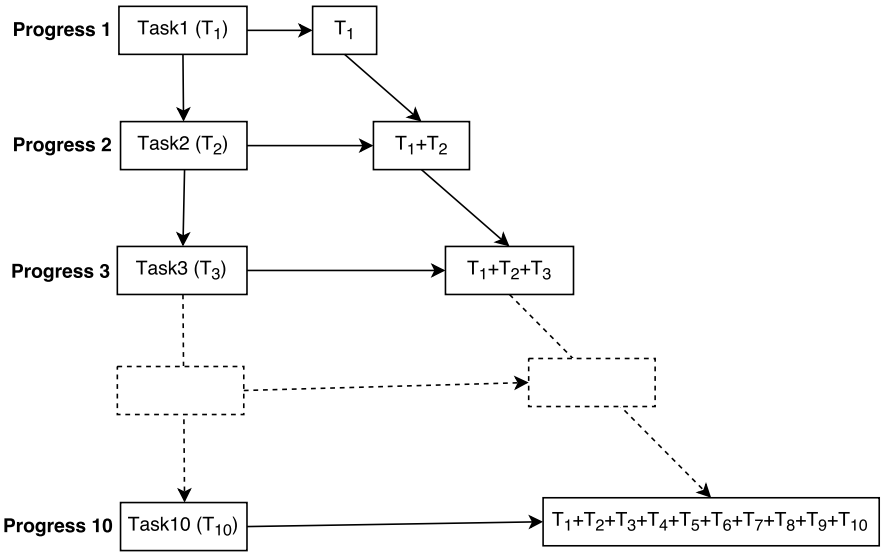

Figure 3.4: Work progress based on tasks completion

As depicted in figure **3.4**, each partial job is a task like Task1, Task2, upto Task10. Total size of file to be processed will be divided into 10 parts and each part will be processed in subsequent steps. If 5^{th} step is running in some point of time and the spot instances got terminated for some reason, only 5^{th} step has to be reprocessed again but not step 1 to step 4 and then can step forward from 6^{th} step and ahead. So the progress is in the rate of 10% for completing each step. This setup should even be beneficial in case of failure in master node itself causing cluster to terminate or core nodes too because the processed results are already in persistent S3 storage. The extra time would be cluster setup and configuration time which is about 5 to 10 minutes for EMR cluster with core Hadoop.

3.6 Bidding Strategies

When bidding for spot instances in Amazon Web Services, the lowest granularity value allowed is $G = 0.001$. In the paper written by Voorsluys and Buyya [47], five possible bidding strategies for spot instance at Amazon Web Serveces EC2 are mentioned. They are minimum, mean, on-demand, high and current price for bidding.

Two bidding strategies will be implemented in this project based on median of prices of given duration and current or latest price. Formula for both of these strategies are similar. The first one based on median of prices will be formulated as below:

$$B = M + (D - M) \cdot x$$

where B = bidding price, M = Median of prices in certain duration, D = On-demand instance price, x is increment percent
The second one based on latest price will be formulated as below:

$$B = L + (D - L) \cdot x$$

where B = bidding price, L = Latest spot market price, D = On-demand instance price, x is increment percent

The lower value of x will be referred as Normal bidding while for aggressive bidding, the higher value of x will be chosen.

Part II

Results and Conclusion

Chapter 4

Results I - Design

This chapter discusses algorithms developed to achieve the objectives. Two algorithms were formulated to address the goal of designing and developing the tools that will minimize the cost of batch jobs processing in EMR cluster. Both of the algorithms focused in leveraging spot instances to minimize the cost of the processing job in the cluster for timely completion of job.

The first algorithm **Deadline Aware Auto Bidding Scaling Algorithm (DAAB)** was based on adjusting task capacity in each time slots using spot instances only until the threshold time. After the threshold time, on-demand instances would be used in order to complete tasks within deadline. The bidding strategy chosen for this algorithm is based on median of prices from history. The formula can be referred from section **3.6**.

While the second algorithm **Deadline Aware Progress Adaptive Burst Bidding Algorithm(DPB)** followed the notion of burst bidding of spot instances i.e. using more spot instances than required to finish job before time which also decrease the probability of using on-demand instances after threshold time because the processing would finish before threshold time. The bidding strategy chosen is based on latest spot price. Normal and aggressive bidding is also implemented in this algorithm at defined checkpoints. When the progress at checkpoint is *Behind*, the aggressive bidding will be employed. While if progress at checkpoint is *Ahead* or *In-pace*, the bidding will be normal. The normal and aggressive bidding differs only by increment percent x. Normal bidding strategy has value of $x = 0.02$ while aggressive one has value $x = 0.05$.

4.1 Deadline Aware Auto Bidding Scaling (DAAB) Algorithm

In this algorithm, bidding will be done with $(N_x - \beta)$ on the basis of estimated number of instances N_x for spot instances at the beginning. β is number of core instances running in the cluster which also contributes in task so it is subtracted. The bid price would be sum of median of last 10 hours price and 2 % increment in difference between on-demand price and the median. Then at every 10% of time to deadline from beginning, it will check progress (which will be between 1 and 10) and recalculate new number of instances Z_x required to complete the whole processing in time so $(Z_x - \beta)$ instances will be bid. Bidding will be done with the same bidding price calculated before so task instances will be resized with $(Z_x - \beta)$. This will continue till 90% of time to deadline and at that point new Z_x is recalculated and new $(Z_x - \beta)$ will be count for all on-demand instances only to guarantee the work completion in time.

Algorithm 1: Deadline Aware Auto Bidding Scaling (DAAB)

Data: Get ClusterID, Cluster-zone, estimated number of instances as $\mathbf{N_x}$, Deadlinetime $\mathbf{t_d}$, β is number of core instances and get median spot market price M from 10 hours history, B refers to bid price and D is on-demand price;

1 Initialize t=0;
2 BidPrice $B = M + (D - M) \cdot 0.02$;
3 $Bid(N_x - \beta, B)$;
4 Resize task instances with N_x spot instances if bid accepted ;
5 **while** $t < 0.9 \cdot t_d$ **do**
6 **foreach** $t = t + 0.1 \cdot t_d$ **do**
7 Find Progress P;
8 RemainingTime $t_r = t_d - t$;
9 $\mathbf{Z_x} = (((10 - P) \cdot N_x \cdot t_d)/(10 \cdot t_r)) - \beta$;
10 $Bid(Z_x, B)$;
11 Resize taskinstances with Z_x spot instances for bid price ;
12 **end**
13 **end**
14 Find Progress P;
15 Calculate $Z_x = ((10 - P) \cdot N_x) - \beta$;
16 Resize taskinstances with Z_x ondemand instances ;

During every 10% of time, if processing get delayed either due to spot instances unavailability or abrupt termination in between, then it will compensate by resizing instances capacity at the start of next time slot. Even if processing was faster than expected which is checked at start of new time slot, it will decrease the size of instances capacity for next 10% of time to keep in pace with expected line. If bid is successful at every time slot and the allocated instances do not get terminated in any of these

time slots till 90% of time, then it is the best case which will be in pace to base-line progress from start to end. The worst case could be no bid successful till 90% of time and the cluster has to be provisioned with only on-demand instances. It will need 10 times of required task instances estimated beforehand. The other worst possibility could be successful provision of spot instances but termination before every 10% progress or unsuccessful bid till the end.

4.1.1 Expected Results of DAAB Algorithm

The illustration for expected perfect case and worst case is depicted in figure **4.1**. It is to be noted that the graph will be somehow different as there will always be two core nodes running which also contribute in performing tasks causing some progress all the time. The other cases between perfect and worst case scenario would be performing total processing work with lesser number of on-demand instances than in the worst case scenario and higher number of spot instances.

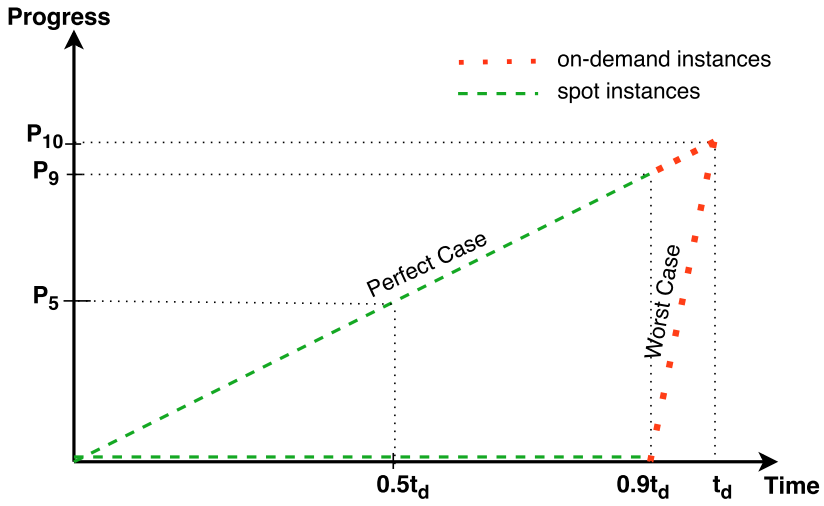

Figure 4.1: Perfect and worst case scenario in Algorithm **1**

4.2 Deadline Aware Progress Adaptive Burst Bidding (DPB) Algorithm

In this algorithm, only spot instances will be used upto 50% of time. The multiplication factor of α enables the cluster to process data as earlier as possible with spot instances. There will be two checkpoints for evaluating progress based on elapsed time and re-adjust number of spot instances

based on latest price and/or on-demand instances to meet deadline.

Algorithm 2: Deadline Aware Progress Adaptive Burst Bidding (DPB)

Data: α is multiplication factor, β is number of core instances
Set ClusterID, Cluster-zone, Instance-number $\mathbf{N_x}$, Deadlinetime $\mathbf{t_d}$;
Find latest spot market price L, t=0 ;
BidPrice $B = L + (D - L) \cdot 0.02$ where $D \rightarrow ondemand\_price$;
$Bid(\alpha \cdot N_x - \beta, B)$;
Resize task instances with $(\alpha \cdot N_x - \beta)$ spot instances for bid price ;
while $t < 0.5 \cdot t_d$ **do**
    foreach $t = t + 0.1 \cdot t_d$ **do**
        if *previous* $(\alpha \cdot N_x - \beta)$ *spot instances still running* **then**
            Continue
        else
            Update BidPrice $B = L + G$;
            Resize task instances with $(\alpha \cdot N_x - \beta)$ spot instances for new bid price B ;
        end
    end
end
if $ProgressP <= 4$ **then**
    Bid for $(\alpha \cdot N_x - \beta)$ spot instances with $B = L + (D - L) \cdot 0.05$ and N_x on demand;
else
    if $P == 5$ **then**
        Bid for $(\alpha \cdot N_x - \beta)$ spot instances with $B = L + (D - L) \cdot 0.05$;
    else
        if $P > 5$ **then**
            Bid for $(\alpha \cdot N_x - \beta)$ spot instances with $B = L + (D - L) \cdot 0.02$;
        else
            Readjust all task instances size to zero and exit;
        end
    end
end
while $t < 0.8 \cdot t_d$ **do**
end
if $P <= 7$ **then**
    Use $((((10 - P)/2) \cdot N_x) - \beta)$ ondemand instances + (N_x) spot instances with $B = L + (D - L) \cdot 0.05$
else
    Use $(N_x - \beta)$ ondemand + N_x spot instances with $B = L + (D - L) \cdot 0.05$
end

These checkpoints will be set at 50% of time elapsed to deadline

and 80% of time elapsed to deadline. Also till 50% of time to deadline, for every 10% of time, status of $(\alpha \cdot N_x - \beta)$ spot instances will be checked and rebidding will be done if they are terminated for some reason otherwise same spot instances will be running. The value N_x is estimated number of task instances required from estimation phase and β is number of core instances running.

At 50% of elapsed time i.e. **Checkpoint 1**, depending on progress different strategies will be chosen for further processing and similar is the case at 80% of elapsed time. As depicted in figure **3.4**, total work will be divided into 10 tasks by dividing equal number of files of same size for each task. So progress bar can be measured in terms of 10% increment like P_0 refers to 0%, P_1 refers to 10%, P_2 means 20% and goes upto P_{10} as 100% progress.

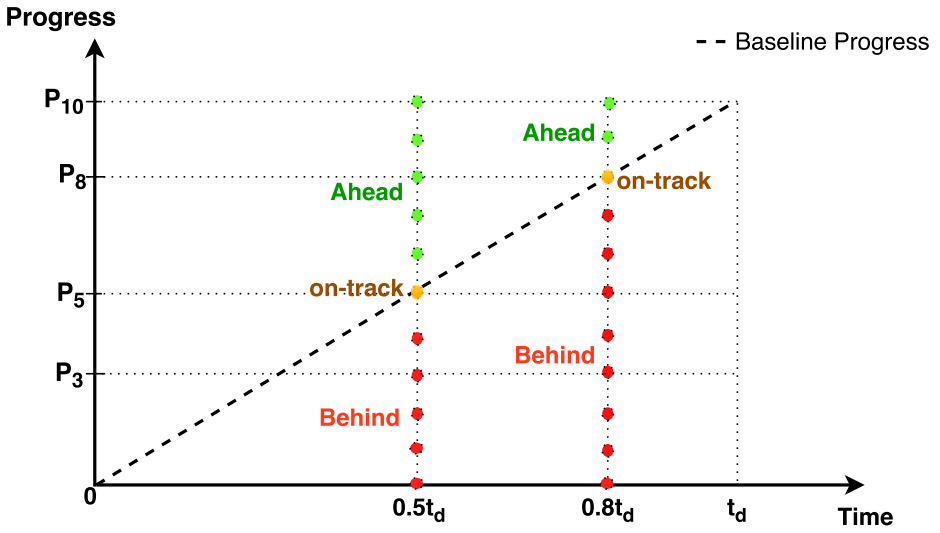

Figure 4.2: Baseline Progress and Checkpoints for Algorithm **2**

On the basis of progress at these checkpoints, further strategies will be applied. The diagram **4.2** depicts the status label at each checkpoint as "Behind" points colored red, "On-track" points colored yellow and "Ahead" points colored green. Strategies are set on the basis of these statuses. There are two bidding strategies viz., **Bid_A** where bid price is sum of latest price and 2% of difference between ondemand price and latest price; and **Bid_B** where bid price is sum of latest price and 5% of difference between on-demand price and latest price. The bidding strategies are mentioned in section **3.6**.

At **Checkpoint 1**, if progress status is *Behind* that means progress is less than 50% then **Bid_B** will be applied for $(\alpha \cdot N_x - \beta)$ spot instances along with N_x on-demand instances in order to improve progress against

baseline progress. If it is *on-track*, $(\alpha \cdot N_x - \beta)$ spot instances with **Bid_B** will be employed in order to finish work ahead of time. While in case of *Ahead*, **Bid_A** will be employed for $(\alpha \cdot N_x - \beta)$ spot instances.

At 80% of time to deadline i.e. **Checkpoint 2**, if its *Ahead* or *on-track*, $(N_x - \beta)$ on-demand instances along with N_x spot instances with **BidB** would be used. While in case of *Behind*, required number of on-demand instances will be calculated as $((((10 - P)/2) \cdot N_x) - \beta)$ along with N_x spot instances to finish up ahead of deadline. On-demand instances guarantees the completion of processing in time while adding spot instances to them speed up processing if they are available till the end.

4.2.1 Expected Results of DPB Algorithm

As depicted in figure **4.3**, perfect case could be getting spot instances for all 50% of time to deadline when $\alpha = 2$. The whole processing should be finished at this point with $(\alpha \cdot N_x - \beta)$ spot instances along with always running β core instances. While worst case could be unavailability of spot instances or termination of spot instances before every 10% completion of work so the progress would be *Behind* at 50% of time and then deploy N_x on-demand instances along with spot instances. However at this point and beyond also spot instances would not be allocated so total processing will be done by on-demand instances only.

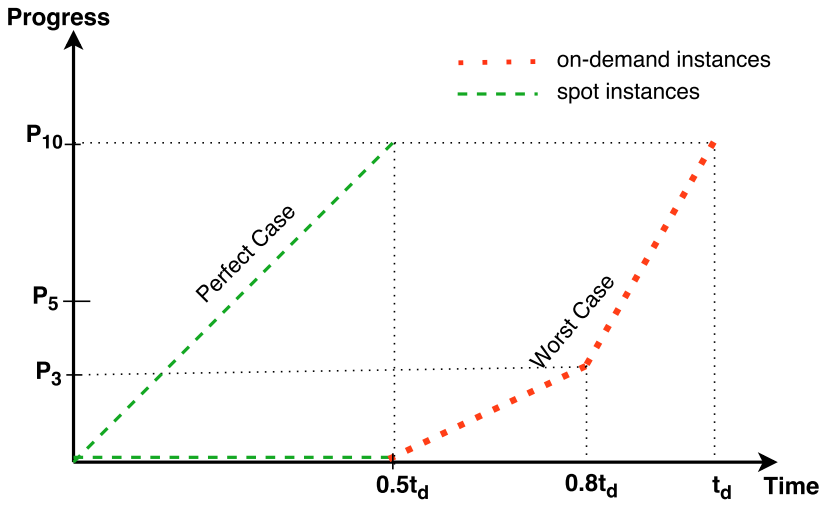

Figure 4.3: Perfect and worst case scenario in Algorithm **2**

This case can even be worse than using all on-demand instances. It is possible that spot instances started processing but terminated and contribute 0% to output. However these spot instances will contribute to cost for their partial existence. The possibility for the worst case is very low because this algorithm employ burst bidding and also bid price is based on

the latest spot market price for each bidding. The diagram could be little different as some progress might have happened at 50% time to deadline as there are always running two core instances which also contribute in processing.

Chapter 5

Results II - Implementation and Experiments

In this chapter, the data outcomes from the experiments based on proposed solutions in Approach Chapter are summarized along with system details, important code snippets and implementation. The first section of this chapter presents the System Setup with developed scripts followed by actual experiments in second and third section. The results from implementing the algorithms **DAAB - Algorithm 1** and **DPB - Algorithm 2** are summarized in the third section **5.4** of this chapter.

5.1 The System Setup

5.1.1 Setting up Boto3 with AWS configuration

It is required to set up authentication credentials in the client machine where python code code is run integrating Boto3. First a new user "kabeen" was created in Identity and Access Management - IAM [40] console of Amazon web service to deploy required IAM roles for Elastic MapReduce [41]. *AmazonElasticMapReduceFullAccess*, *AmazonS3FullAccess* and *AmazonElasticMapReduceforEC2Role* roles were attached to the user with policies in order to get access to the resources required for automation script in python driven by the user "kabeen". *AmazonElasticMapReduceFullAccess* allows the user to do action like *cloudwatch:**, *ec2:RunInstances*, *ec2:RequestSpotInstances*, *elasticmapreduce:**, etc.

AWS Command Line Interface (CLI) was first installed in the local machine. Then access key ID and secret access key was checked from web IAM console. User "kabeen" was already created by clicking on button ***Create Access Key*** in *Security Credentials* tab. After security credentials

had been downloaded securely, it was configured in default credential file "~/.aws/credentials".

```
[default]
aws_access_key_id = XXXXXXXXXXXXXXXXXXX
aws_secret_access_key = XXXXXXXXXXXXXXXXXXX
```

Default region was set as "us-west-2" which is Oregon region and has three availability zones namely us-west-2a, us-west-2b and us-west-2c in file "~/.aws/config".

```
[default]
region = us-west-2
[profile kabeen]
region = us-west-2
```

Boto3 was installed via **pip** [34]:

```
$ pip install boto3
```

5.1.2 Provisioning and running Amazon EMR Cluster

Amazon EMR Cluster is used for analytics. Amazon EMR served data processing with the use of MapReduce log analyzing engine running in Hadoop. For all the experiments, the grant amount from Amazon for Amazon Web Services was utilized. It was applied in AWS programs for Research and Education [13]. The latest release 4.6.0 was used for the experiments with Core Hadoop 2.7.2 as application. The default hardware configuration was 1 master node and 2 core nodes and all instances are of type "m1.medium". The task nodes were adjusted as per computational needs in experiments. The default zone chosen was "us-west-2" which consists of availability zones "us-west-2a", "us-west-2b" and "us-west-2c". For accessibility of the instances that constitute the cluster, a key pair "EMR_test" was generated and used throughout all experiments.

For creating a new cluster each time following code was run for creating a cluster with emr release 4.6.0 and core Hadoop 2.7.2:

Listing 5.1: Provisioning/running EMR with Hadoop - runEMRCluster.py

```
import boto3
from datetime import datetime

emrclient = boto3.client('emr')
response = emrclient.run_job_flow(Name="mycluster01",
  LogUri='s3://aws-test-kabin01/logs',
  ReleaseLabel="emr-4.6.0",
  Instances={'MasterInstanceType':'m1.medium',
    'SlaveInstanceType':'m1.medium',
    'InstanceCount':3,
    'Ec2KeyName':'EMR_test',
```

```
12      'KeepJobFlowAliveWhenNoSteps':True
13    },
14    Configurations=[{"Classification": "emrfs-site",
15      "Properties": {"fs.s3.consistent": "true"}}
16    ],
17    JobFlowRole="EMR_EC2_DefaultRole",
18    ServiceRole="EMR_DefaultRole"
19  )
```

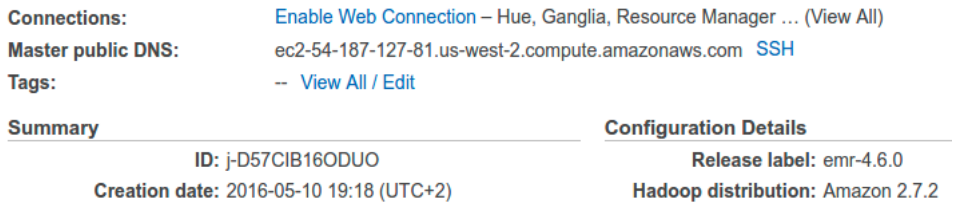

Figure 5.1: EMR Software Info

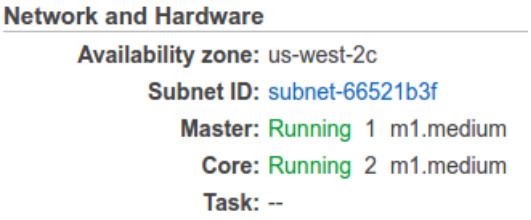

Figure 5.2: EMR Availability Zone and Hardware Info

By running python code as in listing **5.1**, cluster got provisioned and ran in Amazon Web Services with assigned Master public DNS, ClusterID and configuration details as Release Label - emr4.6.0, Hadoop distribution 2.7.2, availability zone as us-west-2b and provisioned and later running master and core instances information. It can be referred from figure **5.1** and figure **5.2** snipped from Amazon EMR Web GUI. The default zone configured was us-west-2 (Oregon) so whenever a cluster was created it starts and run in one of the three availability regions (us-west-2a, us-west-2b and us-west-2c) [38] in that zone.

For viewing nodes status, cluster progress native Hadoop Web GUI were also observed. For this, Foxyproxy was setup in Chrome browser and *foxyproxy-settings.xml* file was created with content as mentioned in the EMR official website [18]. Then by clicking "Use proxies based on their pre-defined patterns and priorities" in foxyproxy symbol, it was activated.

From local machine the following command was fired:

```
ssh -i ./EMR_test.pem -ND 8157 hadoop@ec2-54-191-137-97.us-west-2.
    compute.amazonaws.com
```

Then in the browser, masters public dns address was used along with port 8088 for Hadoop Default Web Interface and port 50070 for Hadoop Datan-

odes status. Amazon EMR has Cloudwatch too which is very convenient for programmers to use with Boto3 Python and thus cloudwatch metrics for Amazon EMR were used in automation scripts. The following are screenshots from Hadoop Web Interface at port 8088 and also from port 50070 about namenodes when six task instances including two core nodes were running in EMR cluster. An application was also tested.

Rack	Node State	Node Address	Node HTTP Address
/default-rack	RUNNING	ip-172-31-30-47.us-west-2.compute.internal:8041	ip-172-31-30-47.us-west-2.compute.internal:8042
/default-rack	RUNNING	ip-172-31-19-190.us-west-2.compute.internal:8041	ip-172-31-19-190.us-west-2.compute.internal:8042
/default-rack	RUNNING	ip-172-31-26-244.us-west-2.compute.internal:8041	ip-172-31-26-244.us-west-2.compute.internal:8042
/default-rack	RUNNING	ip-172-31-30-137.us-west-2.compute.internal:8041	ip-172-31-30-137.us-west-2.compute.internal:8042
/default-rack	RUNNING	ip-172-31-19-189.us-west-2.compute.internal:8041	ip-172-31-19-189.us-west-2.compute.internal:8042
/default-rack	RUNNING	ip-172-31-20-140.us-west-2.compute.internal:8041	ip-172-31-20-140.us-west-2.compute.internal:8042

Figure 5.3: Running task nodes in EMR cluster

Node	Last contact	Admin State	Capacity	Used	Non DFS Used	Remaining	Blocks	Block pool used	Failed Volumes	Version
ip-172-31-19-190.us-west-2.compute.internal:50010 (172.31.19.190:50010)	1	In Service	399.27 GB	35.64 GB	7.46 GB	356.17 GB	303	35.64 GB (8.93%)	0	2.7.2-amzn-1
ip-172-31-19-189.us-west-2.compute.internal:50010 (172.31.19.189:50010)	1	In Service	399.27 GB	37.99 GB	9.33 GB	351.95 GB	320	37.99 GB (9.52%)	0	2.7.2-amzn-1

Figure 5.4: "m1.medium" core nodes in EMR cluster

The figure **5.3** depicts the status of running nodes. It was captured when there were two core nodes and four task nodes were running. In the figure **5.4**, the detailed information about core nodes were rendered and these core nodes represent data nodes of the Hadoop Cluster. A completed application's status was observed in Hadoop GUI in figure **5.5**.

User:	hadoop
Name:	Log Analyzer
Application Type:	MAPREDUCE
Application Tags:	
YarnApplicationState:	FINISHED
FinalStatus Reported by AM:	SUCCEEDED
Started:	Wed May 18 13:20:54 +0000 2016
Elapsed:	1hrs, 9sec
Tracking URL:	History
Diagnostics:	

Figure 5.5: A completed application status in Hadoop GUI

5.1.3 Input Data Generating Script

Following bash script named ***loggenerate.sh*** was developed which generated syslog alike data. Two positional arguments are number of iteration and output file.

```
#!/bin/bash
dt=`date +"%b %d %T"`
hn=`hostname`
num=0
while [ $num -lt $1 ]
do
echo "$dt $hn " "pqrprocess[3689]:" "Info Msg2" >> $2
echo "$dt $hn " "abcprocess[7125]:" "Info Msg3" >> $2
echo "$dt $hn " "mnoprocess[8123]:" "Info Msg4" >> $2
echo "$dt $hn " "uvxprocess[7125]:" "Info Msg5" >> $2
num=$[$num+1]
done
```

Running above bash as ***loggenerate.sh 1100000 samplesyslog01*** give 1100000 lines of sample syslog which was about 100 MB in size and was compressed with bzip2 and compressed file was about 385 KB in size. The default split size in hadoop 2 is 64 MB and supports compressed files like bzip2 and gzip. Compressed file could have been made as big as 64MB even if the file is unsplittable like gzip. However it would require significanly very large processing time for "m1.medium" instances to process each split because of its huge size. Therefore, for the experiments small files were created though it is not recommended for real world data processing.
Few lines of output file are as below:

```
Apr 15 17:03:13 kabin-lappy xyzprocess[6789]: Info Msg 1
Apr 15 17:03:13 kabin-lappy pqrprocess[3689]: Info Msg 2
Apr 15 17:03:13 kabin-lappy abcprocess[7125]: Info Msg 3
Apr 15 17:03:13 kabin-lappy mnoprocess[8123]: Info Msg 4
Apr 15 17:03:13 kabin-lappy uvxprocess[7125]: Info Msg 5
Apr 15 17:03:13 kabin-lappy xyzprocess[6789]: Info Msg 1
```

5.1.4 Creating Custom JAR as MapReduce Application

The source codes for Mapper, Reducer and Driver which analyzed syslogs and count number of events per second are used from this link [19]. These classes were archived as a Java Archive (JAR) file. Each classes represent map, reduce and driver procedures. The main entry point is the driver procedure that joins together the Job flow application and coordinate with MapReduce to use map and reduce tasks programmed in Mapper and Reducer classes respectively. The java files taken from here [19] were *LogMapper.java*, *LogReducer.java* and *LogAnalysisDriver.java*. They were compiled into java class files and later packaged those classes into JAR file.

```
$ javac LogMapper.java LogReducer.java LogAnalysisDriver.java

$ jar -cvf log-analysis.jar LogMapper.class LogReducer.class
    LogAnalysisDriver.class
```

As log analyzer *log-analysis.jar* was ready, it could be used for processing data in map reduce paradigm as figure **3.1**. The log analyzer was uploaded into script folder in "aws-test-kabin01" S3 bucket as below. Refer subsection **2.1.4.3** for more about S3 bucket.

```
import boto3

s3 = boto3.resource('s3')

data = open('log-analysis.jar', 'rb')
s3.Bucket('aws-test-kabin01').put_object(Key='log-analysis.jar',
    Body=data)
```

The loganalyzer *log-analysis.jar* was tested in Amazon EMR to process some syslog type files generated and it was successfully processed and output was saved in S3 storage. The testing was done with EMR Web GUI by adding Step for execution with all required arguments Driver, input location and output location after pointing location of *log-analysis.jar*. This testing was for estimation itself presented with data in subsection **5.2.3**, the result of which would be used in automation script for doing actual data processing. Adding steps can be done programatically or with AWS Command Line Interface Command as well which was done for the main experiments as the detailed command is published at **A.7** in Appendix. The GUI based steps addition is shown as GUI snippet in figure **5.6** below.

5.1.5 Python Scripts for the implementation of algorithms

On the basis of the two algorithms **DAAB Algorithm** and **DPB Algorithm** in Approach chapter, two python scripts were created with functions for specific tasks. Specific tasks were getting cluster metrics, getting latest spot market price, bidding, adding on-demand or spot task nodes, resizing task

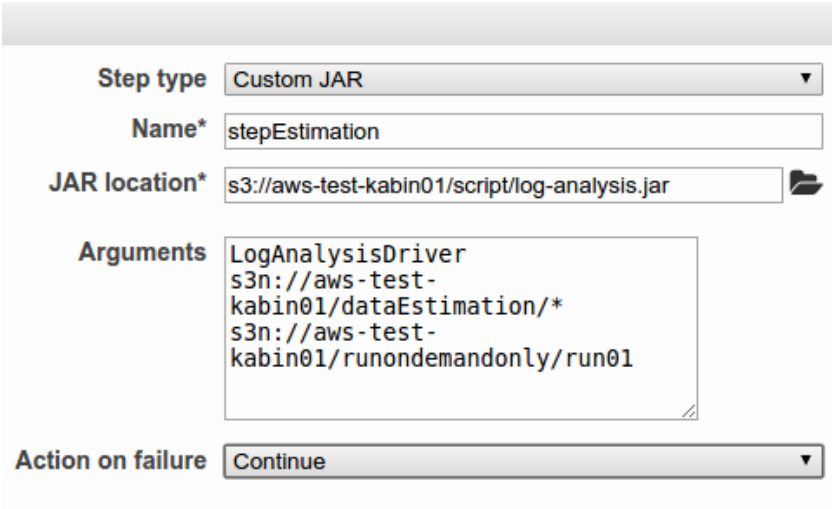

Figure 5.6: Sample Files Processing Step for Estimation

instance group. The main() function has the algorithmic logic which calls these functions with parameters as per requirement. Boto3 python apk for AWS was integreated with python for code development. The detailed fully functional codes for both algorithms are published in listing **A.4** and **A.5** in Appendix. Different functional modules are explained below.

5.1.5.1 Getting Spot Market Price From Historical Data

Amazon Web Services can be asked for historical data of spot market price for previous 90 days from current time. Using boto3, python code were developed for extracting 90 days data as well as for few of latest data. The detail code for 90 days historical data is included in **A.1** in Appendix. For **DPB** algorithm proposed in Approach Chapter, the latest spot market price was implemented in code. Median was calculated from data of latest hour for Algorithm **1** while for Algorithm **2** the latest spot market price was used as in function below.

Listing 5.2: Function for getting latest spot market price

```
1  def getspotpricehistoryfrom(productdesc, availabilityzone,
       instancetypes, frommminutes):
2    response = ec2client.describe_spot_price_history(
3    DryRun = False,
4    StartTime = datetime.datetime.utcnow() - datetime.timedelta(
       minutes=frommminutes),
5    EndTime = datetime.datetime.utcnow(),
6    ProductDescriptions = productdesc, #list e.g. ['Linux/UNIX']
7    AvailabilityZone = availabilityzone, #e.g. 'us-west-2b'
8    InstanceTypes = instancetypes # list of instance types e.g. ['m1
       .medium']
9    )
10
```

```
11    return round(float(response['SpotPriceHistory'][0]['SpotPrice'])
        , 4)]
```

5.1.5.2 Bidding function

Three types of bidding function were made. In **DAAB** algorithm, bid price was to set as sum of median of latest hour's prices and 2% of difference between price of on-demand and median price. While in **DPB** algorithm, for exactly same formula median would be replaced by latest price in **Bid_A** strategy while 5% instead of 2% to add up with latest price in **Bid_B** strategy. Bidding function for **Bid_A** is given in listing below.

Listing 5.3: Function for bidding with **Bid_A** for Algorithm **2**

```
1 def applybidA(count, zone, ClusterID):
2   latestprice = getspotpricehistoryfrom(['Linux/UNIX'], zone, ['m1
      .medium'], 60)
3   bidprice = round(latestprice + (0.087-latestprice)*.02, 3) #
      0.087 is on-demand m1.medium price
4   if bidprice <= 0.6*0.087:
5     addtasknodes('Task_spot', 'spot', 'm1.medium', count, str(
      bidprice), ClusterID)
```

Some bidding tests were done with random bidding price. At one point in time when the spot market price reaches $0.0197, different spot status were observed as in figure **5.7** below. Each spot request had one Request ID in form "sir-XXXXXXXX" and the requested instance type was "m1.medium" with AMI ID "ami-4c1c847c" for "Linux/Unix" Operating System. The initial state and status when spot requested was "open" and "pending-evaluation" respectively. Later on the basis of change of market price different status were observed. Like for bidding price $0.017 in third row in figure **5.7**, the request was previously fulfilled and became active but when price increased, it was terminated. The state "active" and "fulfilled" meant the spot instance request accepted and the instance was spawn with active instance ID too. The status "az-group-constraint" was observed when there already had task group in Amazon EMR Cluster with spot bidding price greater than the newly quoted price.

5.1.5.3 Adding Task Nodes and resize task nodes

As per our algorithms there could be need of either adding spot instances or on-demand instances or even both as task nodes as per requirement at different time slots and checkpoints assumed in the algorithms. Task node named as *Task_spot* or *Task_ondemand*, market as "spot" or "ondemand", instance type like "m1.medium", number of instances required as count, bid price as spotprice and Cluster ID as ClusterID can be passed into function as functional parameters below. It add task nodes into the running

Max Price	Instance	State	Status
$0.015		open	az-group-constraint
$0.02	i-c21be21f	closed	instance-terminated-by-price
$0.017	i-8d12eb50	closed	instance-terminated-by-price
$0.019	i-b90bf264	closed	instance-terminated-by-price
$0.015		cancelled	canceled-before-fulfillment
$0.017		cancelled	canceled-before-fulfillment
$0.021	i-cd768f10	active	fulfilled
$0.021	i-b06c956d	active	fulfilled
$0.021	i-376c95ea	active	fulfilled

Figure 5.7: Spot Requests and Status

cluster. Similarly there is need of task nodes resizing function when the task node group was already created and running in cluster and required to be resized based on condition.

Listing 5.4: Function for adding on-demand or spot instances as task nodes

```
import boto3

emrclient=boto3.client('emr')

def addtasknodes(name, market, insttype, count, spotprice,
    ClusterID):
  if market == "ondemand":
    response = emrclient.add_instance_groups(
    InstanceGroups=[
    {
      'Name': name,
      'Market': 'ON_DEMAND',
      'InstanceRole': 'TASK', #'MASTER'|'CORE'|'TASK'
      'InstanceType': insttype, #e.g. m1.medium
      'InstanceCount': count,
    }
    ],
    JobFlowId=ClusterID
    )
      elif market == "spot":
    response = emrclient.add_instance_groups(
    InstanceGroups=[
                {
      'Name': name,
      'Market': 'SPOT',
      'InstanceRole': 'TASK', #'MASTER'|'CORE'|'TASK'
      'BidPrice': spotprice,
      'InstanceType': insttype,
      'InstanceCount': count,
    }
    ],
    JobFlowId=ClusterID
```

```
32      )
```

5.1.5.4 Get Cluster Metrics

To check the cluster health status, to analyze cluster performance and cluster progress, EMR metrics can be observed using Amazon CloudWatch. The metrics mentioned in subsection 2.2.2.1 were measured using the function developed. One of the important metrics for making decisions in **DAAB** algorithm and **DPB** algorithm is *AppsCompleted*. As mentioned about progress, progress is in step of 10% as total job is divided into ten steps. Completion of each step means there is increase by 1 in *AppsCompleted* metrics. Ihe developed code was checked by executing small jobs in steps. The metrics is updated once every five minutes as EMR cluster sends these metrics every five minutes to the CloudWatch.

Listing 5.5: Function for getting latest metrics value

```
1  cwclient = boto3.client('cloudwatch')
2
3  def getclustermetrics(ClusterID, metricsname, timeinseconds):
4    response = cwclient.get_metric_statistics(Namespace = "AWS/
       ElasticMapReduce",
5      MetricName=metricsname,
6      StartTime = datetime.datetime.utcnow() - datetime.timedelta(
       seconds=timeinseconds),
7      EndTime = datetime.datetime.utcnow(),
8      Dimensions=[{'Name': 'JobFlowId', 'Value': ClusterID},],
9      Period = 60,
10     Statistics=['Average']
11   )
12   metricsvaluelist = sorted(response['Datapoints'], key=itemgetter
       ('Timestamp'), reverse=True)
13   if metricsvaluelist:
14           return int(newlist[0]['Average'])
15       else:
16           return -1
```

5.2 Initial Experiments

5.2.1 Input Data Generattion

Using the bash script mentioned in subsection **5.1.3**, input syslog like logs were generated. 5100 sample files were created and each of 510 files were saved in ten folders named 01, 02 and so on as 10 for the last folder. They were uploaded into *data* folder in S3 bucket labelled "aws-test-kabin01" using AWS S3 Web GUI as in figure **5.8** below. Each of 10 folders contains 10 percent share of total input data to be processed.

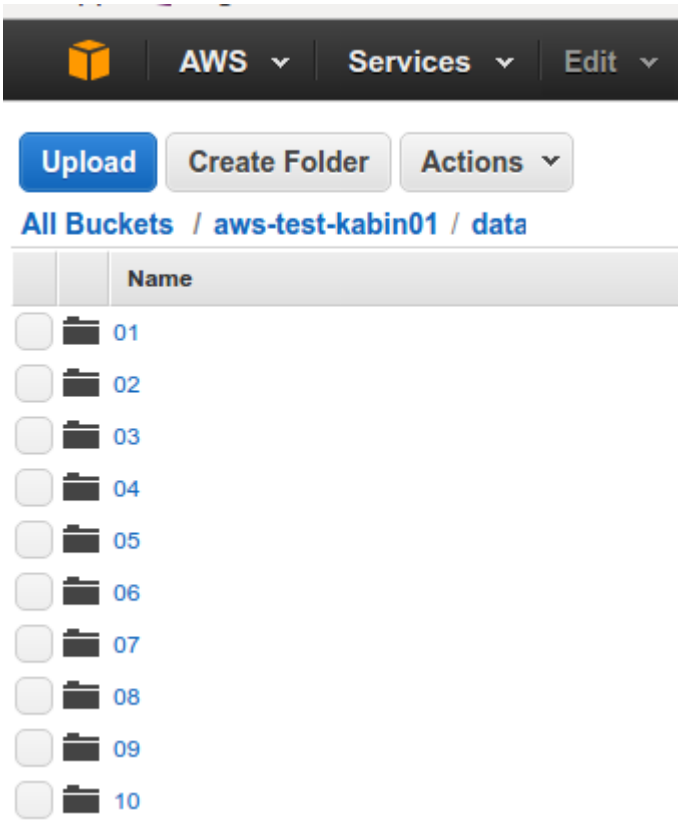

Figure 5.8: Uploading data to S3 bucket from Web GUI

5.2.2 Provision and run the EMR cluster with steps

During setup of EMR cluster at **5.1.2**, the cluster was created, provisioned and run with 1 master instance and 2 core instances. The detailed python codes and Amazon Web Services CLI command are presented in listings **A.6** and **A.7** respectively. When the cluster was created with AWS CLI command as in listing **A.7** in Appendix, the EMR cluster gets provisioned and run with unique cluster ID.

```
"InstanceGroupType":"CORE","InstanceType":"m1.medium","Name":"Core
Instance Group"},{"InstanceCount":1,"InstanceGroupType":"MASTER",
"InstanceType":"m1.medium","Name":"Master Instance Group"}]'
--region us-west-2
{
    "ClusterId": "j-V6WWSMHYCRP7"
}
```

Besides creating, provisioning and running the cluster, the steps were added. Each step would processe 10% of total data to be processed. The files were already uploaded into ten folders equally divided in each folder. In each folder, 510 files were uploaded out of total 5100 files.

The provisioning and running of new EMR cluster was done each time considering it as "Transient Cluster" [16]. It took **7 minutes** to provision, configure and run the EMR cluster with 1 master "m1.medium" instance and 2 core "m1.medium" instances. For all the experiments, it took between **7 minutes** to **10 minutes** for running the EMR cluster with fully configured Core Hadoop 2.7.2.

5.2.3 Estimation Phase

Two sample files were uploaded into *dataEstimation* folder in *aws-test-kabin01* S3 bucket and were processed. The EMR cluster was run with 1 master and 1 core "m1.medium" instance. Using the formula mentioned in subsection **3.4.4**, number of instances required to process all data in deadline was calculated. As discussed in subsection **5.1.3**, files compressed in bzip2, size less than 64MB, would be processed later. This would need 5100 mappers to process all the files. For all the experiments deadline time would be taken as 10 hours. Refer to figure 5.6, the estimation step was added to the cluster using EMR Web GUI.

The formula from subsection **3.4.4** is given by,

$$N_x = \frac{S_m \cdot t_m}{N_m \cdot t_d}$$

In this formula, the calculation was done for estimating number of "m1.medium" instances and it has capacity to run "2" mappers in parallel.

Three experiments were carried out to process 8 sample files. It took 322 seconds, 325 seconds and 332 seconds respectively. The average of three 322.3 seconds was noted down. Deadline time was assumed to be 10 hours i.e. 36000 for all experiments. Calculation was done with $S_\mathrm{m} = 5100$, $t_\mathrm{m} = 327.6$, $N_\mathrm{m} = 2$, $t_\mathrm{d} = 36000$. The calculated value should be divided by "4" because according to the formula, the number of files to be processed should be equal to number of parallel mappers in the instance. 8 files i.e. 4 times mapper capacity was processed in order to get better average. Number of required instances was given by the formula as below.

$$N_x = \frac{5100 \cdot 322.3}{2 \cdot 36000} \cdot \frac{1}{4} = \mathbf{5.7}$$

Rounding up to immediate higher integer gave 6 instances as requirement. For all the experiments $N_\mathbf{x} = \mathbf{6}$ was used as required number of task instances. It is to be noted that as core instances also contribute as task instances so depending on number of core instances running, the value should be subtracted from it. The following log was generated while running the estimation step. The last line gave the execution time of 348 seconds and initial setup time of 28 seconds was subtracted to get 320 seconds as one of obervations in above calculation. The execution time could be observed from the last line of this controller log in listing below.

Listing 5.6: Controller log of Step Execution

```
2016-04-28T23:23:19.267Z INFO Ensure step 2 jar file s3://aws-test
    -kabin01/script/log-analysis.jar
2016-04-28T23:23:19.591Z INFO StepRunner: Created Runner for step2
INFO startExec 'hadoop jar /mnt/var/lib/hadoop/steps/s-11
    Z7RSAEV8JF0/log-analysis.jar LogAnalysisDriver s3n://aws-test-
    kabin01/dataEstimation/* s3n://aws-test-kabin01/runEstimation2
    /run0'
INFO Environment:
  TERM=linux
   ... ...
   ... ...
INFO Working dir /mnt/var/lib/hadoop/steps/s-11Z7RSAEV8JF0
INFO ProcessRunner started child process 13526 :
hadoop   13526  2295  0 23:23 ?        00:00:00 bash /usr/lib/
    hadoop/bin/hadoop jar /mnt/var/lib/hadoop/steps/s-11
    Z7RSAEV8JF0/log-analysis.jar LogAnalysisDriver s3n://aws-test-
    kabin01/dataEstimation/* s3n://aws-test-kabin01/runEstimation2
    /run0
2016-04-28T23:23:23.657Z INFO HadoopJarStepRunner.Runner: startRun
    () called for s-11Z7RSAEV8JF0 Child Pid: 13526
INFO Synchronously wait child process to complete : hadoop jar /
    mnt/var/lib/hadoop/steps/s-11Z7RSAE...
INFO waitProcessCompletion ended with exit code 0 : hadoop jar /
    mnt/var/lib/hadoop/steps/s-11Z7RSAE...
INFO total process run time: 362 seconds
2016-04-28T23:29:23.920Z INFO Step created jobs:
    job_1463353543856_0001
2016-04-28T23:29:23.921Z INFO Step succeeded with exitCode 0 and
    took 348 seconds
```

5.3 Setting up Base Experiment - EMR Cluster with all on-demand instances

Before carrying out the experiments on the proposed algorithms **DAAB** - Algorithm **1** and **DPB** - Algorithm **2**, a benchmark was set by running the cluster with on-demand instances only. This experiment resulted in time and cost reference for the experiments implementing these algorithms. These were the experiments on cluster with task nodes as on-demand instances only and named as **OD-n**. Two experiments were carried out for the base experiment. They were **OD-1** and **OD-2**.

5.3.1 Experiment OD-1 and OD-2

Two experiments **OD-1** and **OD-2** were carried out in with same sets of resources and processing needs. In fact the processing needs and deadline for all the experiments including this one is same. The sample files were created with 1100000 lines and compressed in bzip2 format. The size details

can ne referred in section **5.1.3**.
Processing requirements for the experiments were as following:

Total number of files to process is 5100
Number of files to process per step is 510
$Deadline = 10\ hours$

The required number of instance for this processing was estimated as $5.7 \approx 6$. So the EMR cluster was provisioned and run with following hardware configuration having on-demand instances only.

Instance Flavor was "m1.medium" for all instances.
Number of master instances was 1
Number of core instances was 2
Number of task instances was 4

Two core instances also contribute as task nodes so total task capacity became 6 which is as per estimation. There were two clusters with same configuration and the clusters started to process data. All the experiments were done in Amazon's Oregon Zone called "us-west-2" which had three availability zones "us-west-2a", "us-west-2b" and "us-west-2c".

5.4 Experiments on the Algorithms DAAB and DPB

The two algorithms Deadline Aware Auto Bidding Scaling **(DAAB)** and Deadline Aware Progress Adaptive Burst Bidding **(DPB)** were implemented as developed python scripts. Number of experiments were made on scripts based on **DAAB** and **DPB** algorithms. Number of experiments were carried out for both algorithms. The experiments are represented in the form of **DAAB-n** and **DPB-n** such that $n = \{1, 2, 3, ...\}$. The processing requirement for these experiments is same as in base experiment in which only on-demand instances were used. They were:

Total number of files to process = 5100
Number of files to process per step = 510
$Deadline = 10\ hours$

On the basis of required number of instances estimated as 6, with algorithm specific adjustment on it data got processed in the EMR cluster. The hardware configuration of EMR cluster in task instances were dynamic and could be only spot-instances or both spot instances and on-demand instances or only on-demand instances in different time slots in each experiment based on the algorithms. The hardware configuration were as below:

Instance Flavor was "m1.medium" for all instances.
Number of master instances was 1 (on-demand instance)

Number of core instances was 2 (on-demand instances)
Number of task instances was dynamic (spot or both or on-demand)

All **DAAB-n** experiments ran for whole time till the deadline while **DPB-n** experiments were shorter than deadline time. The analysis would be made on next chapter on the basis of these experiments against base experiments.

Chapter 6

Results III - Analysis

In the analysis chapter, the results from the experiments are presented comprehensively. There were different experiments carried out as two base experiments with on-demand instances only called **OD-n**, and each of six emperiments as implementation of two algorithms **DAAB** and **DPB** represented as **DAAB-n** and **DPB-n**. Three experiments on DAAB-n and two experiments on DPB-n are presented with detail data and graphs.

As discussed in previous chapter, the instance flavor used was "m1.medium" and details of it can be referred from table **3.1**. Hadoop 2.7.2 was implemented with Amazon EMR release 4.6.0 and operating system type was Linux/Unix with Amazon Machine Image (AMI) ID "ami-44966224".

6.1 Evaluation of base experiments OD-n

The two experiments were carried out as base experiments using task nodes as on-demand instances only. The detail of hardware setup and input files processed is mentioned in subsection **5.3.1**. The following table **6.1** gives individual time for processing each of 10 parts of total data in 10 subsequent steps. Each single part of data processing is represented by $Part - n$ where $n = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 10\}$.

On the basis of estimated instance numbers as "6" rounded of from value "5.7" in subsection **5.2.3**, and deadline time was set as 10 hours so each part the processing time was expected to be less than 1 hour i.e. 3600 seconds. The actual data from experiment OD-1 and OD-2 as in table **6.1** were plotted as boxplots in figure 6.1 to realize consistency of data processing time against each other. Median of data processing time for OD-1 and OD-2 were 3466.5 seconds and 3478 seconds respectively, difference of which was 11.5 seconds which signifies almost equal processing time.

There were no outliers too in both boxplots, so drastic change in processing time was not observed while employing same resources to process each part. While analyzing about individual sets of data from **OD-1** and **OD-2** for processing parts of data, the standard deviation were 108.97 seconds and 87.72 seconds i.e. 3.1% and 2.53% from their mean respectively. If 5% more resources than the value from estimation phase were employed, the data processing would be finished within deadline.

These experiments **OD-1** and **OD-2** were set as benchmark or base experiments as the total data processing completed within deadline using persistent on-demand instances in required numbers as estimated.

Data Processing time in OD-1 and OD-2 (seconds)		
Part	**OD-1**	**OD-2**
Part-1	3370	3425
Part-2	3395	3550
Part-3	3620	3385
Part-4	3553	3501
Part-5	3441	3585
Part-6	3690	3319
Part-7	3415	3507
Part-8	3492	3455
Part-9	3368	3541
Part-10	3495	3368
Total time	**34839**	**34636**

Table 6.1: Data Processing time in OD-n experiments in EMR Cluster

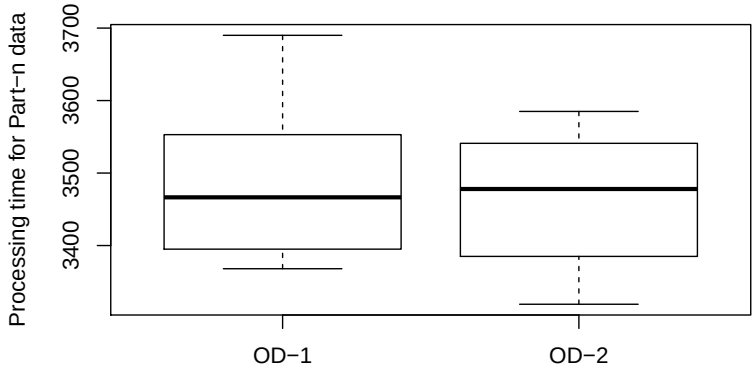

Figure 6.1: Boxplot of data processing time in OD-1 and OD-2

The total cost for each of these two experiments was same. The following table 6.2 summarizes the total cost for running each OD-n experiment with instance flavor "**m1.medium**" and instance operating

system "**Linux/Unix**".

EMR cluster cost for **OD-n** in US dollars					
Instance Type	**Qty**	**Hours**	**EMR Charge/hr**	**Ondemand charge/hr**	**Amount**
master	1	10	0.022	0.087	1.09
core	2	10	0.022	0.087	2.18
task	4	10	0.022	0.087	4.36
Total ($)					7.63

Table 6.2: EMR Cluster cost for **OD-n** experiment

The progress vs time graph for OD-1 is depicted in figure 6.2. Time at every 10% progress were noted and plotted in this graph. The actual processing in OD-1 seemed to be slightly faster than baseline progress (100% processing in exact 10 hours i.e. 600 minutes). Cluster with estimated task nodes as on-demand instances formed a benchmark or base experiment.

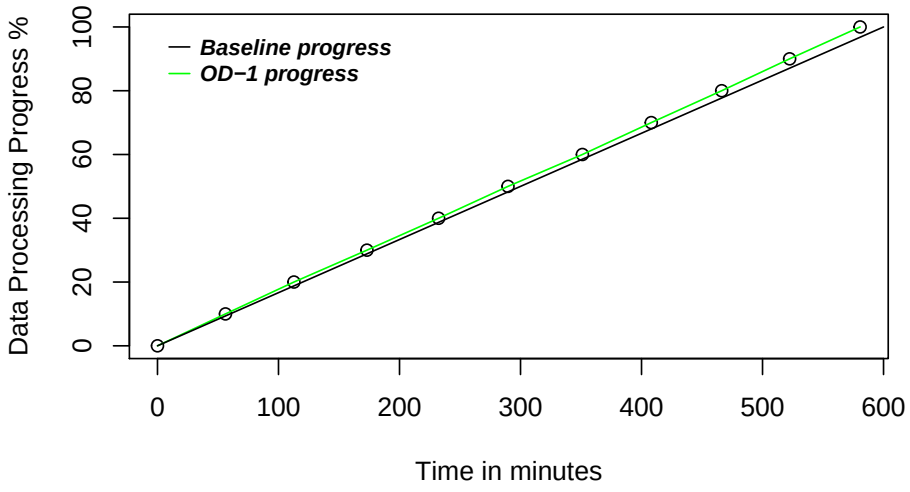

Figure 6.2: Data processing progress vs time in Experiment **OD-1**

6.2 Evaluation of experiments DAAB-n

Ten experiments were carried out for algorithm DAAB. Three experiments are presented in details. Six experiments were not interrupted by spot termination while other four had termination at some point of time. **DAAB-1** experiment is discussed below. There was no spot termination

once the spot instances were allocated in **DAAB-1** and while there were spot instances termination due to increased price named as **DAAB-2** and **DAAB-3**.

6.2.1 DAAB-1 Experiment

The progress vs time graph for DAAB-1 is depicted in figure 6.3

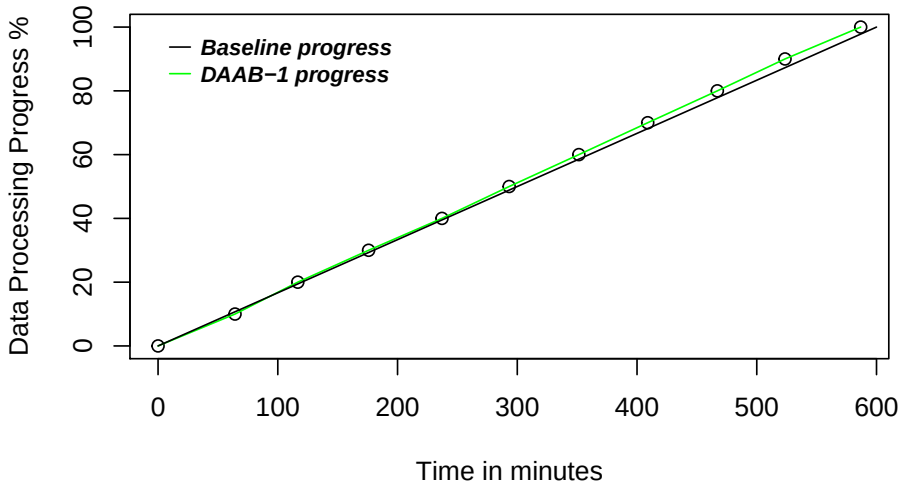

Figure 6.3: Data processing progress vs time in **DAAB-1**

From figure **6.3**, it was the case in which all the spot instances did not terminate throughout cluster lifetime and the progress seemed to be linear with baseline progress. It even finished little earlier than deadline because most of 10% part of processing were processed in less than an hour. As in figure **6.4**, first 4 instances were bid and after bid being fulfilled, number of active mapreduce nodes became 6 as always running 2 core instances were present.

The first part of processing took more than an hour about 64 minutes and 15 seconds, so when task nodes were calculated for remaining processing, it required 7 instances so task capacity was increased by 1. With 7 task nodes the processing of data from 10% to 20% was completed in just 52 minutes and 25 seconds so at second hour adjustment, the task nodes capacity was resized back to 6 and it went on without termination. After 90% of time, only on-demand instances were used to complete remaining processing. As before 90% of time, the last 10% of progress was already started with 2 core instances the work completed before deadline. If the

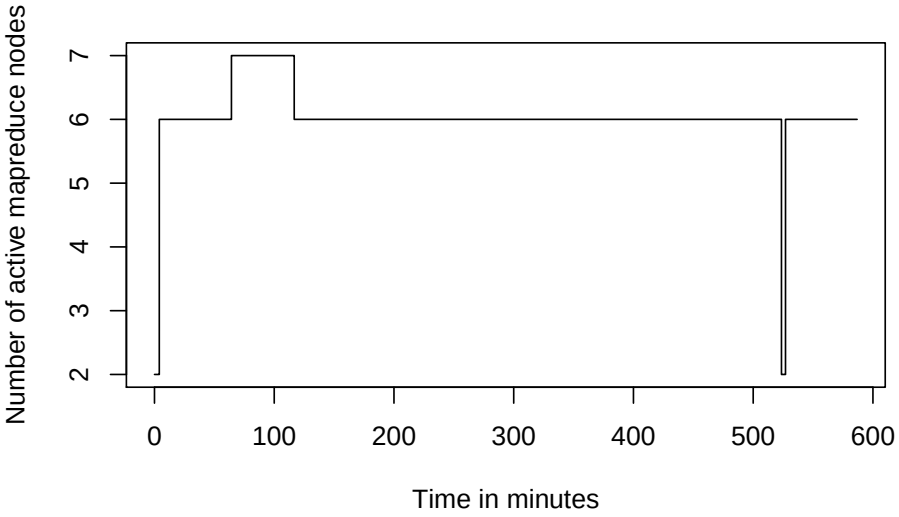

Figure 6.4: Number of running MapReduce nodes in **DAAB-1**

10^{th} of processing started after on-demand instances were running, the processing would have delayed because it takes 4-6 minutes for running new on-demand instances. Better precaution would always be keeping 10% tolerance to deadline. Setting deadline as 90% of actual deadline in automation tool would be an idea.

6.2.2 DAAB-2 Experiment

In this experiment, the spot instances were terminated after 2 hours 3 minutes. That caused to drop mapreduce nodes from 7 to 2 i.e. core instances only and caused the part-3 processing time to 166 minutes and 12 seconds. As spot instances bidding were not successful at 3^{rd} and 4^{th} hour bidding.

The bidding strategy of DAAB-2 was same when first bid calculated from median of 10 hours history. The spot instances were available on the start of sixth hour again. New task instances required became 9 and at beginning of 8^{th} hour it became 10 and ran for two hours. After 9 hours, only on-demand instances were used for processing. However it took 4 minutes 14 seconds more time because of initial setting up of on-demand instances as task nodes to on-demand instances. As talked about in **DAAB-1** experiment, 10% tolerance to deadline would make work completion in time.

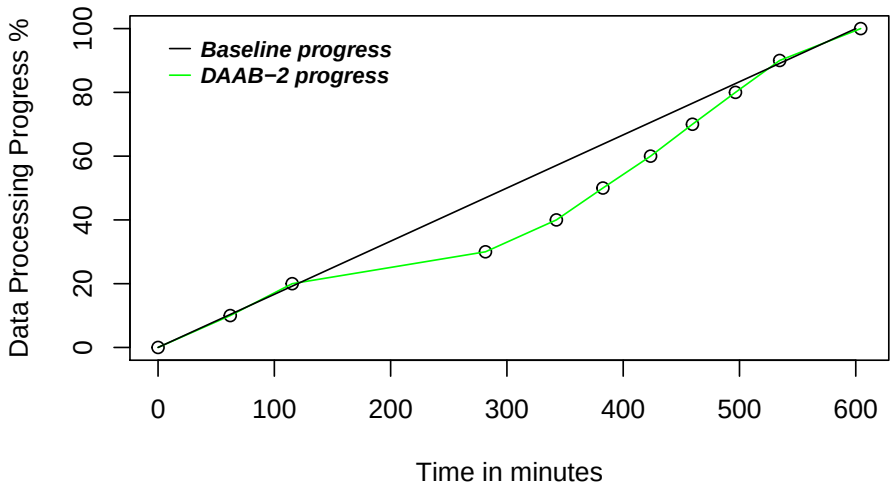

Figure 6.5: Data processing progress vs time in **DAAB-2**

Due to interruption of spot instances for around 3 hours, later spot instances were adjusted to higher values to keep progress in pace with baseline progress.

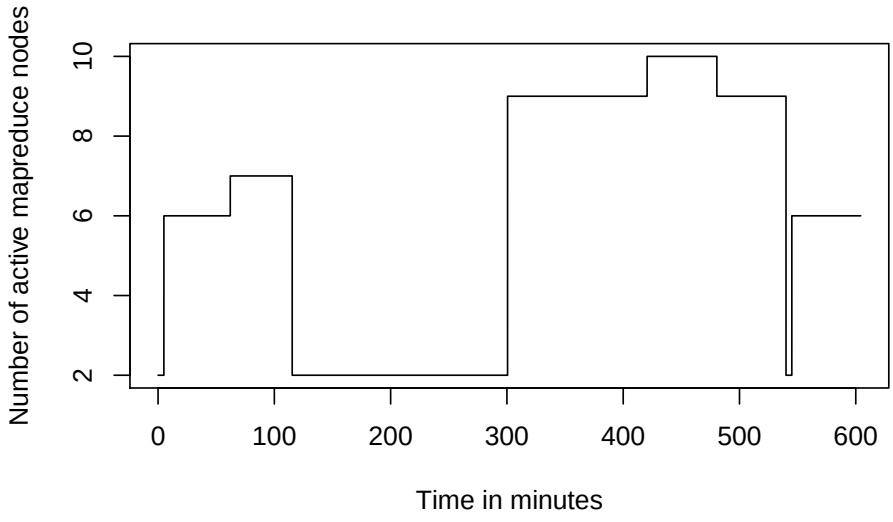

Figure 6.6: Number of running MapReduce nodes in **DAAB-2**

6.2.3 DAAB-3 Experiment

In this experiment, upto 60% of progress, it went on pace with baseline progress. But for the 7_{th} part, it took a long time of 170 minutes because of losing spot instances. At the end, it required 18 instances to complete the processing. 16 on-demand instances as task nodes were deployed after 90% of time. It was noted that when provisioning new instances at 90% of time, cosideration should be made for time the instances took to become up and running. In this case also, it crossed deadline by 4 minutes.

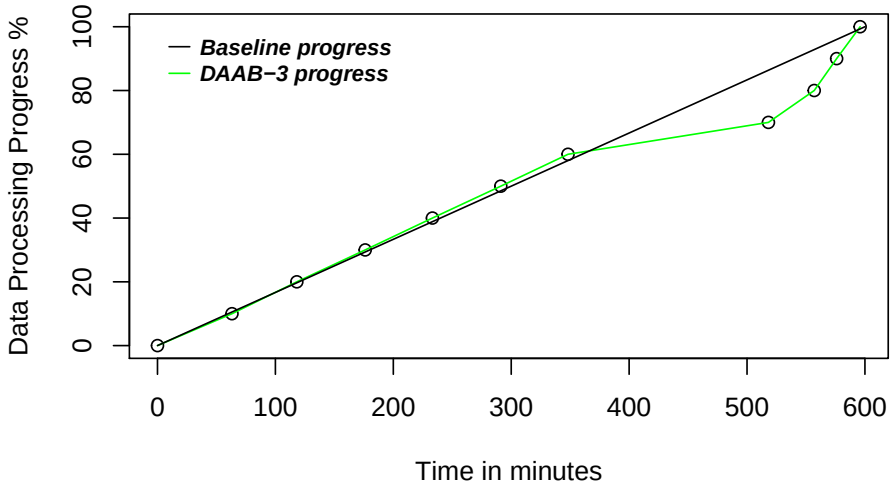

Figure 6.7: Data processing progress vs time in **DAAB-2**

6.2.4 DPB-1 Experiment

In this experiment, as α was set to 2 in automation script. Since bidding did not terminate due to price, it went on smoothly and the total data processing was finished in 290 minutes. **DPB** algorithm uses latest price for bidding. If bid were terminated new bidding would be made with the new calculated price at every 10% of time till 50% of total time. Due to burst factor α, the processing were completed before half of the time.

6.2.5 DPB-2 Experiment

DPB-2 experiment had one time spot termination at time after 2 hours and half hours so the processing was delayed for Part-6 i.e. 50% to 60% data

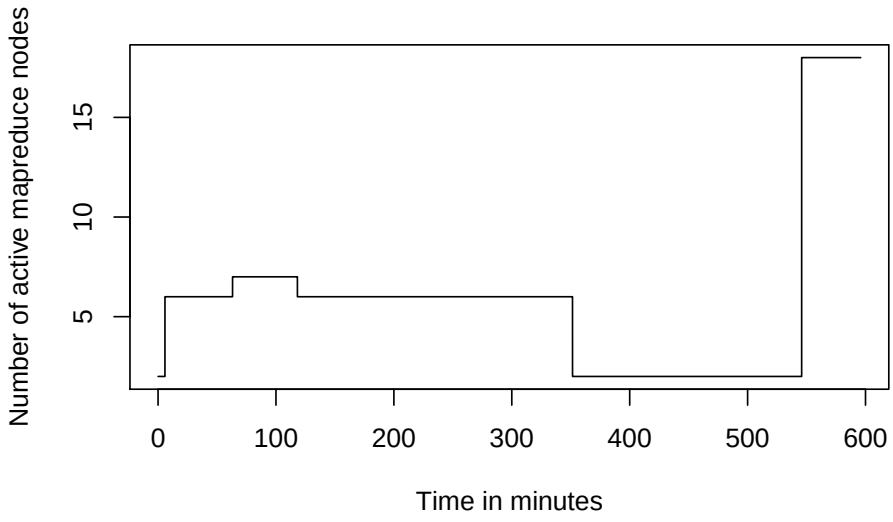

Figure 6.8: Number of running MapReduce nodes in **DAAB-3**

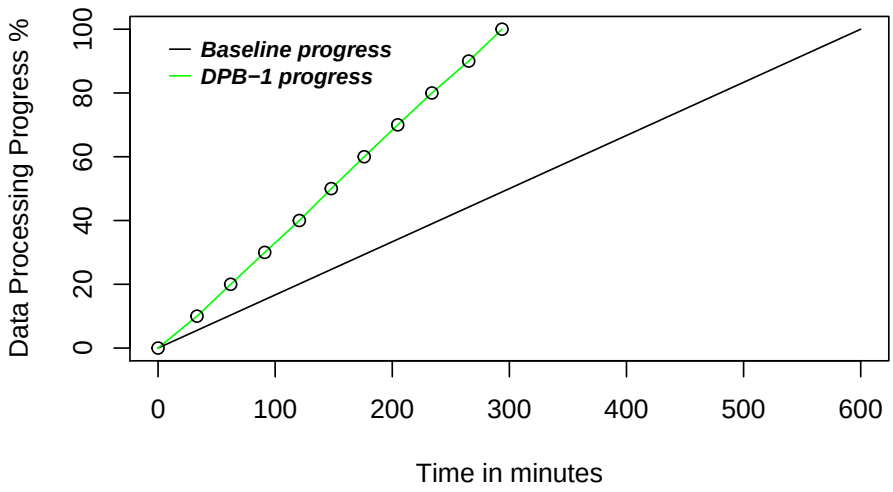

Figure 6.9: Data processing progress vs time in **DPB-1**

which took 1 hour and 5 minutes to process. The number of mapreduce nodes over time for **DPB-2** is depicted in figure **6.11**. New bidding made on 4^{th} hour after 3 hours of processing, the bid got successful with new bidding and again 8 instances were resized as task instances to become

10 map reduce nodes including 2 core nodes. After 50% of time, as the progress was already 90% so as per **DPB** the progress was "Ahead".

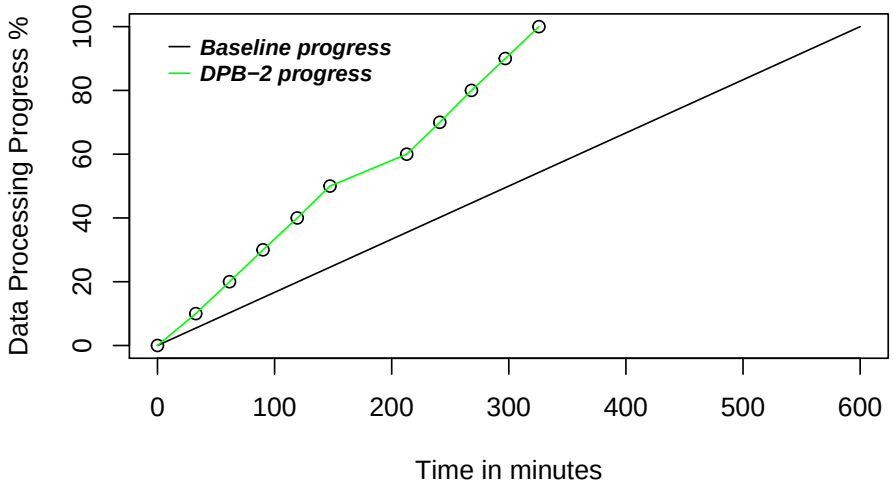

Figure 6.10: Data processing progress vs time in **DPB-2**

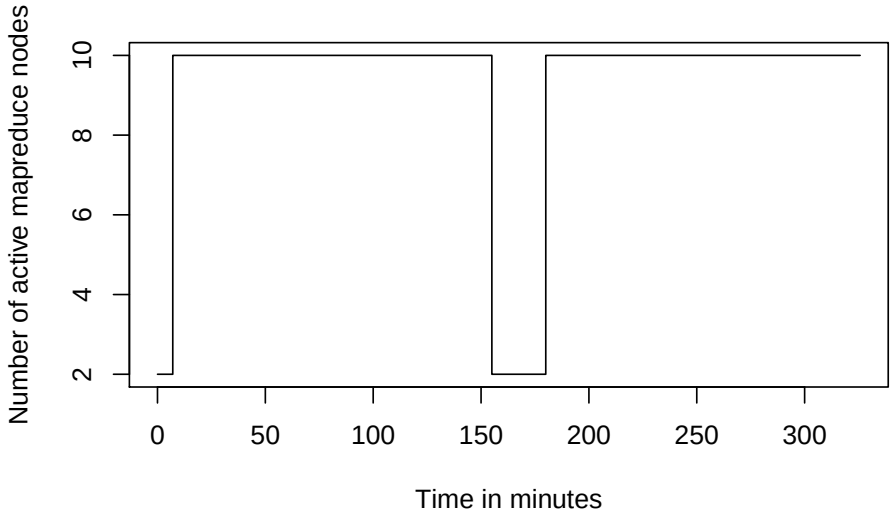

Figure 6.11: Number of running MapReduce nodes in **DPB-2**

As discussed in Approach chapter, data processing would be done in divisions. Figure **6.12** shows the actual data processing status in Amazon

EMR which was extracted from Amazon EMR Web GUI. The divisions *part-n* are represented as steps *Stepn* and each step e.g. Step01 processed part-1, Step02 processed part-2 of data and so on.

ID	Name	Status	Start time (UTC+2)	Elapsed time
s-37IQBUFTXNH91	Step10	Completed	2016-05-15 01:35 (UTC+2)	28 minutes
s-XQ47CP4OPUO2	Step09	Completed	2016-05-15 01:06 (UTC+2)	28 minutes
s-2J127V2B28HWT	Step08	Completed	2016-05-15 00:39 (UTC+2)	27 minutes
s-3PEC5OEN3F3PP	Step07	Completed	2016-05-15 00:11 (UTC+2)	28 minutes
s-1EN9E2F0NPZVN	Step06	Completed	2016-05-14 23:05 (UTC+2)	1 hour, 5 minutes
s-1DT789PMWA604	Step05	Completed	2016-05-14 22:37 (UTC+2)	27 minutes
s-2ZGP2Y1VE5SGZ	Step04	Completed	2016-05-14 22:08 (UTC+2)	29 minutes
s-8N8WD1UOELQ2	Step03	Completed	2016-05-14 21:39 (UTC+2)	28 minutes
s-9WSM7KOD20FP	Step02	Completed	2016-05-14 21:10 (UTC+2)	28 minutes
s-34862D2SSBNGM	Step01	Completed	2016-05-14 20:38 (UTC+2)	32 minutes

Figure 6.12: Screenshot of data processing progress in **DPB-2** from EMR GUI

6.3 Cost Analysis

The cost analysis were made for the experiments presented in this chapter. The total EMR cluster cost and task instances implementation cost were calculated. The total cost can be referred as **OD-n** experiment cost from table **6.2**. This OD-n experiment set benchmark for both performance and cost. Cost of **OD-n** experiment would be the base price for processing batch jobs in this work. On-demand instance charge per hour for "m1.medium" instances is US $0.087. Also there is a charge called EMR charge which is US $0.022 per hour for "m1.medium" instance [11]. Comparing costs on the basis of total EMR cost and task instants' total cost for each experiments were done. The total cluster's cost for each experiment and total task instants's cost for each experiment were recorded in **table 6.3**.

OD-n was referenced as the base experiment. From graphs in figure **6.13** and **6.14**, total emr cost prices and instant task prices can be compared. The total EMR cost was US $7.63, the breakdown of cost can be referred from table **6.2**. This is the base price for the cluster to process the amount of data mentioned in start of the experiment. As task nodes are also on-demand instances, the cost for task nodes is high in OD-n experiments and the price is fixed i.e. US $3.48 as on-demand instance has fixed price. In experiment **DAAB-n** and **DPB-n**, the task instances were either spot instances only or both or on-demand instances at different times. Total EMR cluster price drops by 35%, 34% and 22% in experiments DAAB-1, DAAB-2 and DAAB-3 respectively. In DAAB-1 and DAAB-2, only after

EMR cluster cost and task instants' cost in US $		
Experiment Name	**Total EMR Cost**	**Task instants' Cost**
OD-n	7.63	3.48
DAAB-1	4.99	0.83
DAAB-2	5.07	0.87
DAAB-3	5.94	1.75
DPB-1	3.31	0.58
DPB-2	3.8	0.83

Table 6.3: EMR Cluster cost for all experiments

90% of time, 4 ondemand instances were used as task instances. While in DAAB-3 due to spot instance termination in last few hours caused the need of 16 ondemand instances for last hour processing. This increased the cost considerably.

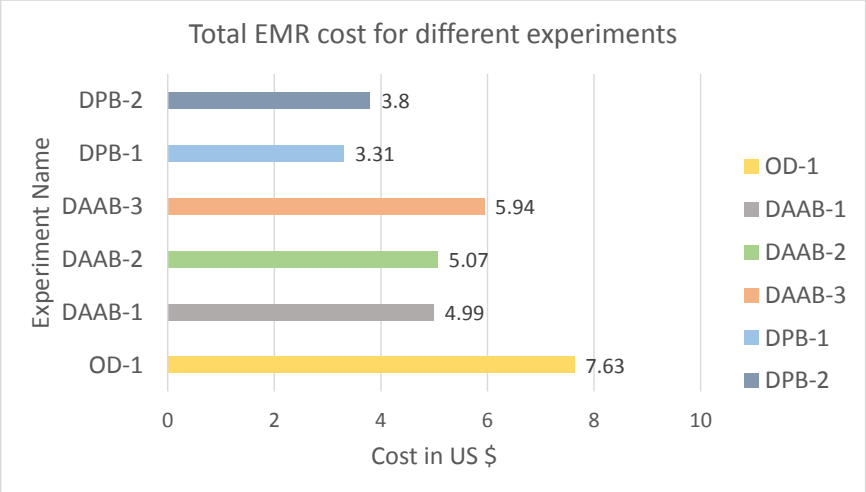

Figure 6.13: Total EMR Cluster price for different experiments

On the other hand, DPB-1 and DPB-2 decreases the cost by 56% and 52% respectively. The huge drop in DPB-1 and DPB-2 is contributed by fast processing also. By around 50% of time, the total processing were completed so the cost for running 1 master and 2 core instances decreased by almost half.

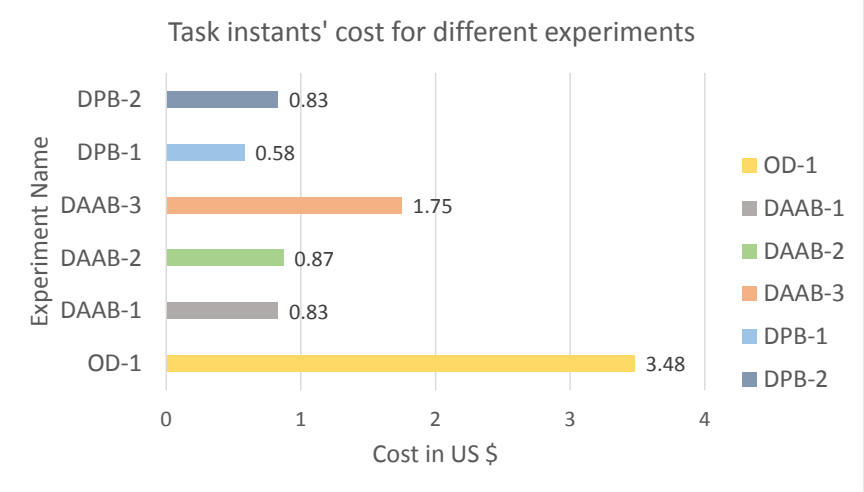

Figure 6.14: Task instances price for different experiments

It is interesting to compare task instances price only. As depicted in figure 6.14, the base price for task instances is US $3.48. For DAAB-1 and DAAB-2 in which only spot instances were used till 90% of time, the price had dropped significantly by 76% and 75% respectively. However, cost for DAAB-3 dropped by only 50% due to need of using large number of spot instances at the end to meet deadline. DPB-1 and DPB-2 performed better as the price was reduced by 83% and 76%. Many experiments were done with DPB and they performed better as they ended up with use of spot instances only and no on-demand instances. During the experiments, the price ranged from US $0.009 to US $0.021 in Oregon (us-west-2) region in Amazon.

The starting spot market price for DAAB-1, DAAB-2 and DAAB-3 were US $0.0134, US $0.0121 and US $0.0132 respectively. While they were US $0.0113 and US $0.0166 respectively for DPB-1 and DPB-2 which changed over time on supply-demand principle of Amazon.

Chapter 7

Discussion

This chapter discusses the implementation steps, facts and challenges along with suggestions for improvements of the developed prototypes as future work of this project.

7.1 Project Evaluation

The huge batch processing can be done in the cluster of commodity hardware. It gives processing power, fault tolerance and low cost. The development of softwares like Apache Hadoop and Apache Spark are making the large-scale data processing convenient. Since the main goal of this project was to minimize the cost of cluster computing on batch jobs, focus was shifted in scaling operations than building tuned clusters using Hadoop or other distributed data processing framework. However, the idea behind Hadoop and map reduce programming paradigm was required. Hadoop installation, configuration and map reduce programming implementation were carried out to process data. The main components, parameters and requirements were known. The prototype implementation was done in Amazon Elastic MapReduce (EMR) as it provides managed Hadoop framework.

The phases in the project were data generation, time estimation and actual data processing. Assumptions were made for deadline time as 10 hours. Another assumption was to use more than 5 task instances for processing total data in 10 hours so that effect on cost and time could be visualized. With more than 5 "m1.medium" instances, it can process huge amount of data. Around 1.9 GB of compressed data in bzip2 format was created. Uncompressed volume was about 500 GB. To create this much data, first Syslog like logs were generated with 1.1 million lines in each file. As hadoop supports compression like bzip2, gzip and LZO, the splittable file format bzip2 was chosen. Each comressed file was of size 380-

390 KB which originally was around 100 MB. 50 files were first generated, compressed and later replicated to generate 5100 files that would processed in EMR.

For better CPU time, smaller file should have been avoided. The file size should be as big as normal hadoop split size of 64 MB if the compressed file is not splittable. The smaller file increases total mapper spawn times as each file will be assigned to individual mapper [20]. In this project, small compressed files were chosen though. It was because low capacity instance "m1.medium" were being used for the experiments. Primary goal was to scale the cluster. Apparently, in this project hadoop performance tuning was not focussed.

Time estimation was done on the basis of formula in subsection **5.2.3**. From the results of data processing time in table **6.1**, standard deviations of 3.1% and 2.53% were observed in two sets of experiments OD-1 and OD-2. Employing tolerance of 5%-10% on deadline time or number of required instances could result in timely completion. When new spot instances are requested at some point of time or new on-demand instances are run, it takes 5-8 minutes to up and run. Hence tolerance is required.

For data processing, the two algorithms **DAAB** and **DPB** were devised. Cost and performance comparision of algorithms **DAAB** and **DPB** are discussed in section **7.2**. Few points were considered while designing both algorithms. Termination is not guaranteed in spot instances and termination notice is very short. When running task instances in spot instances, the intermediate mapper results in these nodes could be lost due to abrupt termination. Hence, data to be processed were divided into ten divisions and processing was done division by division as step execution of mapreduce program. Each step saved the final output into persistent Simple Storage Service (S3) after every division was completed. This meant termination of spot instances could affect a single part at a time. Maximum of 10% data could be affected in each termination. If spot instances were terminated after 70% of data being processed by them it would considerably increase both cost and processing time due to loss of more than 70% of data when division of work was not implemented. The alternative solution for it could be use of snapshot checkpoints which means to create snapshot of instance at regular interval. If the instance gets terminated, new instance will be created from that snapshot. However, there would be need of coordination between Resource Manager and other components in Hadoop.

7.2 Comparison of DAAB and DPB algorithms

The two alogorithms were developed and deployed in python codes. The bidding price should be chosen in such a way that it would not be very high

but should not be smaller than market price. Higher bid price could get better priority but would eventually contribute in increased spot market price. In **DAAB** algorithm, bid price was set as $(M + (D - M) \cdot x)$ where M stands for median price of 10 hours, D stands for on-demand instance price and x is increment percent. This price would be used for bidding from start to the end. The median price was used to avoid possible spikes however it could be the case that bid price calculated at the beginning, would always be lower than market price. Hence it could cause the need of more on-demand instances at the end. In **DPB** algorithm, latest market price was used. The bid price was $(L + (D - L) \cdot x)$ where L represents latest price, D for on-demand instance price and x for increment percent. The advantage of using latest price was observed in **DPB-2** experiment, where the instances were terminated due to price. Later the bid was successful in next round of bidding after the instance was terminated. In **DAAB-3** experiment, no bidding was successful in subsequent bidding once spot instances were terminated at some point in time. Thus it required more number of on-demand instances to complete processing in time.

From the cost analysis in Analysis Chapter, the EMR cluster with all on-demand instances was referenced for other experiments. **DPB-n** experiments seemed to be cheaper than **DAAB-n** experiments. It was because of need of using on-demand instances at the end in **DAAB-n** experiments. Even the case was worse in **DAAB-3** experiment because of need of using more on-demand instances at the end. The worst case for **DPB-n** experiments could also be like worst case for **DAAB-n**. However probability of need of on-demand instances after first checkpoint in **DPB-n** experiments is low due to use of multiplier factor α for required number of instances and implementation of latest price in each new bid. There is high probability that **DPB-n** experiments use only spot instances for task nodes due to use of α. There were around ten experiments for **DPB-n** and all ended up using only spot instances. Two experiments on **DPB-n** were presented with data in previous chapter.

Using these algorithms, the total EMR cost dropped by 22% to 35% in DAAB-n experiments while by almost 50% in DPB-n. If only task nodes were considered, the task capacity price was dropped heavily upto 83%. The cost of all the experiments in this project was covered with Amazon AWS Educate Credits [12].

7.3 Future works

There are many more facets that can be done in this work. In **DAAB** algorithm, latest price can be employed so the probability of bidding to get success in next round would be very high. For both algorithms, the multiplier factor α could be implemented. Depending on the market price, α can be increased or decreased i.e. higher value of α for low spot market

price and vice versa. As the current project saves output in consistent datastore after processing every 10% of data, master and core instances can also be deployed in spot instances. The bidding price for master and core instances can be set higher enough to prevent from being terminated. It can be employed with the concept of division of data and saving of output for each division in persistent storage. Even if master and core instances get terminated due to spot price, new cluster can be provisioned and run in just 7 to 10 minutes with configured Hadoop. So remaining divisions of data can be processed by planning new task capacity.

All the experiments were carried out in three availability zones within Oregon region. Regions other than Oregon (us-west-2) may also be studied and compared.

Besides, the hadoop tuning portion can also be considered. For example, number of mappers can be increased by determining low CPU usages in task nodes. File sizes can also be considered. As discussed above, big file sizes significantly reduce total time in spawning mappers. Few mappers need to be spawned for same amount of data. It is $(x - y) \cdot$ *mapper_spawning_time* time overhead for processing x number of smaller files than y number of bigger files. Core nodes disk capacity should be checked if it is required to process higher voulume of data than core nodes capacity. Different options are available like adding extra volumes to the core instances itself or adding more core nodes. In Amazon EMR, Apache Spark can also be implemented for data processing which is said to be very faster than MapReduce [23]. The same task capacity varying algorithms DAAB and DPB can be used for Spark running in Amazon EMR. Study on peak hours and off hours for spot instances can also be integrated and would be beneficial. Configuring own Hadoop cluster in EC2 instances can save per instance hour EMR charge though it would require extra works on configuring Hadoop, creating custom metrics for monitoring cluster status.

Chapter 8

Conclusion

The objective of this project was to design and develop a tool which can minimize the cost for processing deadline based batch jobs in Amazon cluster leveraging spot instances. Two algorithms were desinged and implemented using Python and boto3 AWS SDK.

First sample data were generated. For the distributed data processing, Amazon EMR with managed Hadoop capability was used. Mapreduce java codes were deployed as Custom Jar in the EMR cluster to process the sample data. To meet the deadline, the algorithms **DAAB** and **DPB** were implemented as automation tool for varying task capacity of Amazon EMR.

DAAB algorithm focused on adapting number of task instances in order to be in pace with baseline progress. After 90% of time, it would vary task capacity with only on-demand instances. While **DPB** was based on bidding α times required instances to complete processing earlier and reducing probability of requiring on-demand nodes at later stages. Another notable difference between **DAAB** and **DPB** was use of bidding strategy. The **DAAB** used medium of last 10 hours for calculating bid price and it was made static during entire data processing life. However in **DPB**, latest market price was used to calcualte bid price and also on termination new bid price was calculated using latest price for required numbe of spot instances. Finally some future works were mentioned. Tuning Hadoop in different ways were also suggested which could result in processing of same amount of data for less time.

Bibliography

[1] *About Amazon EMR Releases - Amazon Elastic MapReduce*. URL: https://docs.aws.amazon.com/ElasticMapReduce/latest/ReleaseGuide/emr-release-components.html (visited on 04/16/2016).

[2] *Amazon EC2 m1.medium - Live Performance Benchmarks - cloudlook*. URL: http://www.cloudlook.com/amazon-ec2-m1-medium-instance#month (visited on 04/11/2016).

[3] *Amazon EC2 Reserved Instances*. URL: https://aws.amazon.com/ec2/purchasing-options/reserved-instances/ (visited on 02/20/2016).

[4] *Amazon EC2 Spot Instances*. URL: https://aws.amazon.com/ec2/spot/ (visited on 02/20/2016).

[5] *Amazon Elastc MapReduce Dimensions and Metrics - Amazon CloudWatch*. URL: http://docs.aws.amazon.com/AmazonCloudWatch/latest/DeveloperGuide/emr-metricscollected.html (visited on 03/11/2016).

[6] *Amazon EMR Release 4.0.0 with New Versions of Apache Hadoop, Hive, and Spark Now Available*. URL: https://aws.amazon.com/about-aws/whats-new/2015/07/amazon-emr-release-4-0-0-with-new-versions-of-apache-hadoop-hive-and-spark-now-available/ (visited on 04/10/2016).

[7] *Amazon Machine Images (AMI) - Amazon Elastic Compute Cloud*. URL: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/AMIs.html (visited on 03/02/2016).

[8] *Amazon Simple Storage Service (S3) - Cloud Storage*. URL: https://aws.amazon.com/s3/ (visited on 03/15/2016).

[9] *Amazon Virtual Private Cloud (VPC) - Amazon Web Services*. URL: https://aws.amazon.com/vpc/ (visited on 03/02/2016).

[10] *Apache Hadoop 2.7.2 - Apache Hadoop YARN*. URL: http://hadoop.apache.org/docs/current/hadoop-yarn/hadoop-yarn-site/YARN.html (visited on 03/10/2016).

[11] *AWS | Amazon Elastic MapReduce (EMR) | Pricing*. URL: https://aws.amazon.com/elasticmapreduce/pricing/ (visited on 03/15/2016).

[12] *AWS Educate*. URL: https://aws.amazon.com/education/awseducate/ (visited on 03/22/2016).

[13] *AWS Programs for Research and Education*. URL: https://aws.amazon.com/grants/ (visited on 03/02/2016).

[14] *AWS Webcast - Amazon Elastic Map Reduce Deep Dive and Best Practices*. URL: http://www.slideshare.net/AmazonWebServices/amazon-elastic-map-reduce-deep-dive-and-best-practices#51 (visited on 03/16/2016).

[15] David Bader. "Cluster Computing: Applications." In: vol. 15. 2. 2001, pp. 181–185.

[16] *Choose the Cluster Lifecycle: Long-Running or Transient - Amazon Elastic MapReduce*. URL: http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-plan-longrunning-transient.html (visited on 03/16/2016).

[17] *Choose the number and type of instances - Amazon Elastic MapReduce*. URL: http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-plan-instances.html (visited on 04/10/2016).

[18] *Configure Proxy Settings to View Websites Hosted on the Master Node - Amazon Elastic MapReduce*. URL: https://docs.aws.amazon.com/ElasticMapReduce/latest/ManagementGuide/emr-connect-master-node-proxy.html (visited on 03/29/2016).

[19] *Data Collection and Data Analysis with AWS - Programming Elastic MapReduce*. URL: https://www.safaribooksonline.com/library/view/programming-elastic-mapreduce/9781449364038/ch02.html (visited on 03/15/2016).

[20] *Deep Dive - Amazon Elastic MapReduce (EMR)*. URL: http://www.slideshare.net/AmazonWebServices/deep-dive-amazon-elastic-map-reduce#29 (visited on 03/24/2016).

[21] *EC2 Instance Types - Amazon Web Services (AWS)*. URL: https://aws.amazon.com/ec2/instance-types/ (visited on 03/10/2016).

[22] *Elastic Computing Cloud (EC2) Cloud Server and Hosting - AWS*. URL: https://aws.amazon.com/ec2/ (visited on 03/01/2016).

[23] *FAQ | Apache Spark*. URL: http://spark.apache.org/faq.html (visited on 04/01/2016).

[24] *Global Infrastructure of Amazon AWS*. URL: https://aws.amazon.com/about-aws/global-infrastructure/ (visited on 03/06/2016).

[25] Google. *Search Engine Strategies Conference*. 2006. URL: http://www.google.com/press/podium/ses2006.html (visited on 02/02/2016).

[26] Jared Gray and Thomas C Bressoud. "Towards a MapReduce Application Performance Model." In: *Midstates Conference*. 2012.

[27] Xin He et al. "Cutting the cost of hosting online services using cloud spot markets." In: *Proceedings of the 24th International Symposium on High-Performance Parallel and Distributed Computing*. ACM. 2015, pp. 207–218.

[28] *Instance Groups - Amazon Elastic MapReduce*. URL: http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/InstanceGroups.html (visited on 03/05/2016).

[29] Eric Knorr. *What cloud computing really means | Infoworld*. 2008. URL: http://www.infoworld.com/article/2683784/cloud-computing/what-cloud-computing-really-means.html (visited on 02/03/2016).

[30] Shen Li et al. "WOHA: Deadline-aware Map-Reduce workflow scheduling framework over Hadoop clusters." In: *Distributed Computing Systems (ICDCS), 2014 IEEE 34th International Conference on*. IEEE. 2014, pp. 93–103.

[31] Lakshay Malhotra, Devyani Agarwal, and Arunima Jaiswal. "Virtualization in Cloud Computing." In: *J Inform Tech Softw Eng* 4.136 (2014), p. 2.

[32] Ishai Menache, Ohad Shamir, and Navendu Jain. "On-demand, spot, or both: Dynamic resource allocation for executing batch jobs in the cloud." In: *Proc. of USENIX International Conference on Autonomic Computing*. 2014.

[33] *Overview - Spark 1.6.1 Documentation*. URL: http://spark.apache.org/docs/latest/ (visited on 04/14/2016).

[34] *pip8.1.2 : Python Package Index*. URL: https://pypi.python.org/pypi/pip (visited on 04/10/2016).

[35] *Presto Documentation - Presto 0.146 Documentation*. URL: https://prestodb.io/docs/current/ (visited on 04/14/2016).

[36] *Purchasing Options - Amazon Web Services*. URL: https://aws.amazon.com/ec2/purchasing-options/ (visited on 03/02/2016).

[37] Antonio Regalado. *Who coined the term "Cloud Computing"*. A Technology Review. URL: http://www.thebusinesstechnologyforum.com/2011/10/who-coined-the-term-cloud-computing (visited on 02/02/2016).

[38] *Regions and Availability Zones - Amazon Elastic Compute Cloud*. URL: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/using-regions-availability-zones.html (visited on 03/04/2016).

[39] *Reports on Critical Capabilities for Public Cloud Infrastructure as a Service Worldwide*. URL: http://www.gartner.com/doc/reprints?id=1-2QQX6UM&ct=151027&st=sb (visited on 03/06/2016).

[40] *Required IAM Roles for Amazon EMR - Amazon Elastic MapReduce*. URL: http://docs.aws.amazon.com/IAM/latest/UserGuide/console.html (visited on 04/20/2016).

[41] *Required IAM Roles for Amazon EMR - Amazon Elastic MapReduce*. URL: http://docs.aws.amazon.com/ElasticMapReduce/latest/DeveloperGuide/emr-iam-roles-req.html (visited on 04/20/2016).

[42] Margaret Rouse. *What is cloud computing - Definition from Whatis.com*. URL: http://searchcloudcomputing.techtarget.com/definition/cloud-computing (visited on 02/03/2016).

[43] Yang Song, Murtaza Zafer, and Kang-Won Lee. "Optimal bidding in spot instance market." In: *INFOCOM, 2012 Proceedings IEEE*. IEEE. 2012, pp. 190–198.

[44] *Spot Bid Status - Amazon Elastic Compute Cloud*. URL: http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/spot-bid-status.html (visited on 03/10/2016).

[45] Supreeth Subramanya et al. "SpotOn: a batch computing service for the spot market." In: *Proceedings of the Sixth ACM Symposium on Cloud Computing*. ACM. 2015, pp. 329–341.

[46] *Top cloud infrastructure-as-a-service vendors*. URL: http://www.cio.com/article/2947282/cloud-infrastructure/top-cloud-infrastructure-as-a-service-vendors.html (visited on 02/24/2016).

[47] William Voorsluys and Rajkumar Buyya. "Reliable provisioning of spot instances for compute-intensive applications." In: *Advanced information networking and applications (aina), 2012 ieee 26th international conference on*. IEEE. 2012, pp. 542–549.

[48] Miles Ward. *Optimizing for Cost in the Cloud*. AWS Summit 2012, NYC. URL: http://www.slideshare.net/AmazonWebServices/optimizing-your-infrastructure-costs-on-aws/22 (visited on 01/31/2016).

[49] *Welcome to Apache Hadoop!* URL: http://hadoop.apache.org/ (visited on 02/05/2016).

[50] *Welcome to Paramiko's documentation - Paramiko Documentation*. URL: http://docs.paramiko.org/en/1.16/ (visited on 04/11/2016).

[51] *What is a Virtual Private Server (VPS)? - Definition from Techopedia*. URL: https://www.techopedia.com/definition/4800/virtual-private-server-vps (visited on 03/10/2016).

[52] *What is big data ? - Definition from Whatis.com*. URL: http://searchcloudcomputing.techtarget.com/definition/big-data-Big-Data (visited on 02/22/2016).

[53] *What is Computer Cluster?* URL: https://www.techopedia.com/definition/6581/computer-cluster (visited on 02/20/2016).

[54] *Why Gartner is Critical to Your Business*. URL: http://www.gartner.com/technology/why_gartner.jsp (visited on 02/24/2016).

[55] Norman Wilde and Thomas Huber. "Virtualization and Cloud Computing." In: *University Of West Florida* (2009).

[56] Noha Xue. "Automated cloud bursting on a hybrid cloud platform." In: (2015).

[57] Liang Zheng et al. "How to Bid the Cloud." In: *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication*. ACM. 2015, pp. 71–84.

Appendices

Appendix A

Developed Python scripts and AWS CLI Commands

Different python scripts developed for the experiments of this thesis are included here. There is also an Amazon Web Service Command Line Interface (CLI) commands for creating EMR cluster and adding steps.

A.1 Python Script to save spot price history of 90 days

This code gets the spot price for last 90 days. Days can be adjusted but Amazon provides maximum 3 months history of spot instance market price.

Listing A.1: Saving Spot market price for 90 days in file

```
__author__ = 'kabin'
#!/usr/bin/python

import boto3
import datetime

ec2client = boto3.client('ec2')

def getpricelist(lstxyz, nxttoken, numdays):
  response = ec2client.describe_spot_price_history(
  DryRun=False,
  StartTime=datetime.datetime.utcnow() - datetime.timedelta(days=
    numdays),
  EndTime=datetime.datetime.utcnow(),
  ProductDescriptions=['Linux/UNIX'],
  AvailabilityZone='us-west-2b',
  InstanceTypes=['m1.medium'],
  NextToken=nxttoken
  )
  nxttoken = response['NextToken']
  print len(response['SpotPriceHistory'])
```

```
  for i in range(0, len(response['SpotPriceHistory'])):
    lstxyz.append(response['SpotPriceHistory'][i]['SpotPrice'])
  if (response['NextToken']!=''):
    getpricelist(lstxyz, nxttoken, numdays)
  return lstxyz

def main():
  a = getpricelist([], '', 90)
  for i in range(len(a)-1, 0, -1):
    with open('spothistory.txt', 'a+w') as test_file:
      test_file.write(str(a[i])+"\n")

if __name__ == '__main__':
  main()
```

A.2 Python Script to get EMR cluster metrics

This script generates all the metrics of Amazon EMR for time period provided in specified file location.

Listing A.2: Getting EMR Cluster metrics using CloudWatch in Boto3

```
__author__ = 'kabin'
#!/usr/bin/python

import os

client = boto3.client('cloudwatch')

def saveclustermetricsinfile(ClusterID, metricsname, timeinseconds
    , filepath):
  response = client.get_metric_statistics(Namespace = "AWS/
    ElasticMapReduce",
    MetricName=metricsname,
    StartTime = datetime.datetime.utcnow() - datetime.timedelta(
    seconds=timeinseconds),
    EndTime = datetime.datetime.utcnow(),
    Dimensions=[{'Name': 'JobFlowId', 'Value': ClusterID},],
    Period = 60,
    Statistics=['Average']
  )
  newlist = sorted(response['Datapoints'], key=itemgetter('
    Timestamp'), reverse=True)

  for i in range(len(newlist)-1, 0, -1):
    with open(filepath+"/"+metricsname, 'a+w') as fileloc:
      fileloc.write(str(newlist[i]['Average']) + " " + str(newlist
    [i]['Timestamp'])[11:16] + "\n")


def main():
  metricslist = ['IsIdle','ContainerAllocated','ContainerReserved'
    ,'ContainerPending','AppsCompleted','AppsFailed','AppsKilled',
    'AppsPending','AppsRunning','AppsSubmitted','CoreNodesRunning'
    ,'CoreNodesPending','LiveDataNodes','MRTotalNodes','
```

```
    MRActiveNodes','MRLostNodes','MRUnhealthyNodes','
    MRDecommissionedNodes','MRRebootedNodes','S3BytesWritten','
    S3BytesRead','HDFSUtilization','HDFSBytesRead','
    HDFSBytesWritten','MissingBlocks','TotalLoad','MemoryTotalMB',
    'MemoryReservedMB','MemoryAvailableMB','MemoryAllocatedMB','
    PendingDeletionBlocks','UnderReplicatedBlocks','
    DfsPendingReplicationBlocks','CapacityRemainingGB']
    ClusterID = 'j-W5ZY30E9E16C'
    directoryname = 'metrics-'+ClusterID+'/metrics'

    for metrics in metricslist:
      if not os.path.exists(directoryname):
            os.makedirs(directoryname)
          saveclustermetricsinfile(ClusterID, metrics, 54000,
    directoryname)

if __name__ == '__main__':
  main()
```

A.3 Python Script to get Map and Reduce Info

This code bundled with Paramiko is used to get Map and Reduce information from EMR master node to the local machine.

Listing A.3: Getting Map and Reduce Info

```
__author__ = 'kabin'
#!/usr/bin/python

import paramiko
import StringIO

key_f = open('./EMR_test.pem','r')
s = key_f.read()
keyfile = StringIO.StringIO(s)
mykey = paramiko.RSAKey.from_private_key(keyfile)
ssh = paramiko.SSHClient()
ssh.set_missing_host_key_policy(paramiko.AutoAddPolicy())
ssh.connect('ec2-54-186-207-237.us-west-2.compute.amazonaws.com',
    username='hadoop', pkey=mykey)
stdin, stdout, stderr = ssh.exec_command('mapred job -status
    job_1462298928419_0001')

listout = stdout.readlines()

InfoDict = {}

InfoDict['NumberofMaps'] = listout[5]
InfoDict['NumberofReduces'] = listout[6]
InfoDict['MapCompletion'] = listout[7]
InfoDict['ReduceCompletion'] = listout[8]
InfoDict['JobState'] = listout[9]
InfoDict['FailureReason'] = listout[11]

print InfoDict
```

A.4 Automation Script 1

This script is implementation of **DAAB** algorithm.

Listing A.4: Automation of **DAAB** Algorithm

```
__author__ = 'kabin'
#!/usr/bin/python

import boto3
import datetime
import paramiko
import StringIO
import time
import math
import sys
from operator import itemgetter


emrclient=boto3.client('emr')
ec2client = boto3.client('ec2')
cwclient = boto3.client('cloudwatch')

#####addtasknode definition#####
def addtasknodes(name, market, insttype, count, spotprice,
    ClusterID):
    #emrclient = boto3.client('emr')
    if market == "ondemand":
        response = emrclient.add_instance_groups(
            InstanceGroups=[
                {
                    'Name': name,
                    'Market': 'ON_DEMAND',
                    'InstanceRole': 'TASK', #'MASTER'|'CORE'|'TASK
    '
                    'InstanceType': insttype, #e.g. m1.medium
                    'InstanceCount': count,
                }
            ],
            JobFlowId=ClusterID
        )
    elif market == "spot":
        response = emrclient.add_instance_groups(
            InstanceGroups=[
                {
                    'Name': name,
                    'Market': 'SPOT',
                    'InstanceRole': 'TASK', #'MASTER'|'CORE'|'TASK
    '
                    'BidPrice': spotprice,
                    'InstanceType': insttype, #e.g. m1.medium
                    'InstanceCount': count,
                }
            ],
            JobFlowId=ClusterID
        )
        #end of #addtasknodes(name, market, insttype, count,
    spotprice, ClusterID):
```

```
#####addtasknode definition#####
#To add tasknodes do following
#addtasknodes('test1', 'spot', 'm1.medium', 10, '0.02', 'j-10
    TZJU39JUNCC')


#####getspotpricehistory#####
def getspotpricehistoryfrom(productdesc, availabilityzone,
    instancetypes, frommminutes): #return list of max, min, mean,
    median and latest values in duration
    response = ec2client.describe_spot_price_history(
        DryRun = False,
        StartTime = datetime.datetime.utcnow() - datetime.
    timedelta(minutes=frommminutes),
        EndTime = datetime.datetime.utcnow(),
        ProductDescriptions = productdesc, #list e.g. ['Linux/UNIX
    ']
        AvailabilityZone = availabilityzone, #e.g. 'us-west-2b'
        InstanceTypes = instancetypes # list of instance types e.g
    . ['m1.medium']
    )

    maxprice = max(response['SpotPriceHistory'])['SpotPrice']
    minprice = min(response['SpotPriceHistory'])['SpotPrice']

    #for mean calculation
    total = 0.0
    numdata = len(response['SpotPriceHistory'])
    for i in range(0, numdata):
        total += float(response['SpotPriceHistory'][i]['SpotPrice'
    ])
        #print float(response['SpotPriceHistory'][i]['SpotPrice'])
    mean = total/numdata

    #for median calculation
    if numdata%2 ==0:
        mid1 = numdata/2
        mid2 = mid1 + 1
        summid1mid2 = float(sorted(response['SpotPriceHistory'])[
    mid1]['SpotPrice']) + float(sorted(response['SpotPriceHistory'
    ])[mid2]['SpotPrice'])
        median = summid1mid2/2
    else:
        mid = (numdata+1)/2
        median = sorted(response['SpotPriceHistory'])[mid]['
    SpotPrice']

    #print [float(maxprice), float(minprice), mean, median]
    return [round(float(maxprice), 4), round(float(minprice), 4),
    round(float(mean), 4), round(float(median), 4), round(float(
    response['SpotPriceHistory'][0]['SpotPrice']), 4)]
    #end of getspotpricehistory (timefrom, timeto, productdesc,
    availabilityzone, instancetypes)
#####getspotpricehistory#####


def getclusterprogress():#returns dictionary with keys as
    Mappercent, reduceperceent, JobCounter, Important EMR Metrics
    pass
```

```
94
95
96  def getclustermetrics(ClusterID, metricsname, timeinseconds):
97      response = cwclient.get_metric_statistics(Namespace = "AWS/
        ElasticMapReduce",
98          MetricName=metricsname,
99          StartTime = datetime.datetime.utcnow() - datetime.
        timedelta(seconds=timeinseconds),
100         EndTime = datetime.datetime.utcnow(),
101         Dimensions=[{'Name': 'JobFlowId', 'Value': ClusterID},],
102         Period = 60,
103         Statistics=['Average']
104     )
105     newlist = sorted(response['Datapoints'], key=itemgetter('
        Timestamp'), reverse=True)
106     if newlist:
107         return int(newlist[0]['Average'])
108     else:
109         return -1
110
111
112 def spotinstancesStatus(ClusterID): #return number of active and
        fulfilled spot instances
113     response = ec2client.describe_spot_instance_requests(
114         DryRun=False,
115         Filters = [
116             {
117               'Name':'availability-zone-group',
118               'Values':[ClusterID],
119             }
120         ]
121     )
122     count = 0
123     for i in range(0, len(response['SpotInstanceRequests'])):
124         if response['SpotInstanceRequests'][i]['State'] == 'active
        ' and response['SpotInstanceRequests'][i]['Status']['Code'] ==
         'fulfilled':
125             count += 1
126     return count
127
128
129 def applybid(count, zone, ClusterID):
130     ondemandprice = 0.087 #for us-west-2 region on-demand m1.
        medium price
131     medianprice = getspotpricehistoryfrom(['Linux/UNIX'], zone, ['
        m1.medium'], 600)[3] #3rd index is for median price for 10
        hours
132     bidprice = round(medianprice + (ondemandprice-medianprice)*
        .02, 3) #0.087 is on-demand m1.medium price and bid price
        allowed should be having 3 digits after decimal
133     if bidprice <= 0.6*0.087:
134         addtasknodes('Task_spot', 'spot', 'm1.medium', count, str(
        bidprice), ClusterID)
135
136
137 def checkmastercorestatus(ClusterID):
138     response = emrclient.list_instance_groups(
139         ClusterId=ClusterID
140         )
```

```
141     runningcoreinstances = 0
142     runningmasterinstances = 0
143 
144     for i in range(0, len(response['InstanceGroups'])):
145         if response['InstanceGroups'][i]['Name'] == 'Core Instance
     Group': #this is a list
146             runningcoreinstances = int(response['InstanceGroups'][
    i]['RunningInstanceCount'])
147         if response['InstanceGroups'][i]['Name'] == 'Master
    Instance Group': #this is a list
148             runningmasterinstances = int(response['InstanceGroups'
    ][i]['RunningInstanceCount'])
149 
150     if runningmasterinstances == 1 and runningcoreinstances == 2:
    #return True if 1 master and 2 cores are running
151         return True
152     else:
153         return False
154 
155 
156 def getnumberofrunningnodes(ClusterID, InstanceGroupName):
157     response = emrclient.list_instance_groups(
158         ClusterId = ClusterID
159     )
160 
161     runninginstancesingroup = 0
162 
163     for i in range(0, len(response['InstanceGroups'])):
164         if response['InstanceGroups'][i]['Name'] ==
    InstanceGroupName:
165             runninginstancesingroup = int(response['InstanceGroups
    '][i]['RunningInstanceCount'])
166 
167     return runninginstancesingroup
168 
169 
170 def resizeinstancegroup(ClusterID, InstanceGroupName, size):
171     response = emrclient.list_instance_groups(
172         ClusterId=ClusterID
173     )
174     #runninginstancesingroup = 0
175     IdList = []
176 
177     for i in range(0, len(response['InstanceGroups'])):
178         #print response['InstanceGroups'][i]
179         if response['InstanceGroups'][i]['Name'] ==
    InstanceGroupName: #this is a list
180             #runninginstancesingroup = int(response['
    InstanceGroups'][i]['RunningInstanceCount'])
181             IdList.append(response['InstanceGroups'][i]['Id'])
182 
183     for Ids in IdList:
184         response1 = emrclient.modify_instance_groups(
185                     InstanceGroups = [
186                         {
187                             'InstanceGroupId': Ids,
188                             'InstanceCount' : size
189                         }
190                     ]
```

```
                    )


def main():

    StartTime = datetime.datetime.now().replace(microsecond=0)
    requestedNx = []

    #INITIAL VALUES AFTER CLUSTER IS RUN AND ESTIMATION IS DONE
    td = 10 #deadline is 10 hours
    k = 2 #number of core nodes
    Nx = 6 #get from estimation
    ClusterID = 'j-2K7D20FPFZAA0' #get after cluster is started
    availabilityzone = 'us-west-2a' #get after cluster is run in
    us-west-2

    appscompletednow = 0

    #At 0% time elapsed - STAGE 1
    print "STAGE 1"
    applybid(Nx-k, availabilityzone, ClusterID) #bid price is
    median of 10 hours price + G
    requestedNx.append(str(Nx-k))
    time.sleep(180)

    Timenow = datetime.datetime.now().replace(microsecond=0)

    while (int((Timenow-StartTime).seconds) < int(td*60*60*0.1)):
    #timeelapsed as 0.1 of td in seconds
        print "WAITING IN LOOP TILL 10% TIME ELAPSED"
        time.sleep(60)
        Timenow = datetime.datetime.now().replace(microsecond=0)

    appscompletednow = getclustermetrics(ClusterID, 'AppsCompleted
    ', int((Timenow-StartTime).seconds))
    remainingapps = 10 - appscompletednow
    remainingtime = 0.9*td

    print "STAGE 2"
    NewNx = (int(math.ceil((float(remainingapps)/9)*Nx))) - k
    resizeinstancegroup(ClusterID, 'Task_spot', NewNx)
    requestedNx.append(str(NewNx))
    time.sleep(180)

    Timenow = datetime.datetime.now().replace(microsecond=0)

    while (int((Timenow-StartTime).seconds) < int(td*60*60*0.2)):
    #timeelapsed as 0.1 of td in seconds
        print "WAITING IN LOOP TILL 20% TIME ELAPSED"
        time.sleep(60)
        Timenow = datetime.datetime.now().replace(microsecond=0)

    appscompletednow = getclustermetrics(ClusterID, 'AppsCompleted
    ', int((Timenow-StartTime).seconds))
    remainingapps = 10 - appscompletednow
    remainingtime = 0.8*td

    print "STAGE 3"
    NewNx = (int(math.ceil((float(remainingapps)/8)*Nx))) - k
```

```
244     resizeinstancegroup(ClusterID, 'Task_spot', NewNx)
245     requestedNx.append(str(NewNx))
246     time.sleep(180)
247
248     Timenow = datetime.datetime.now().replace(microsecond=0)
249
250     while (int((Timenow-StartTime).seconds) < int(td*60*60*0.3)):
        #timeelapsed as 0.1 of td in seconds
251         print "WAITING IN LOOP TILL 30% TIME ELAPSED"
252         time.sleep(60)
253         Timenow = datetime.datetime.now().replace(microsecond=0)
254
255     appscompletednow = getclustermetrics(ClusterID, 'AppsCompleted
        ', int((Timenow-StartTime).seconds))
256     remainingapps = 10 - appscompletednow
257     remainingtime = 0.7*td
258
259     print "STAGE 4"
260     NewNx = (int(math.ceil((float(remainingapps)/7)*Nx))) - k
261     resizeinstancegroup(ClusterID, 'Task_spot', NewNx)
262     requestedNx.append(str(NewNx))
263     time.sleep(180)
264
265     Timenow = datetime.datetime.now().replace(microsecond=0)
266
267     while (int((Timenow-StartTime).seconds) < int(td*60*60*0.4)):
        #timeelapsed as 0.1 of td in seconds
268         print "WAITING IN LOOP TILL 40% TIME ELAPSED"
269         time.sleep(60)
270         Timenow = datetime.datetime.now().replace(microsecond=0)
271
272     appscompletednow = getclustermetrics(ClusterID, 'AppsCompleted
        ', int((Timenow-StartTime).seconds))
273     remainingapps = 10 - appscompletednow
274     remainingtime = 0.6*td
275
276     print "STAGE 5"
277     NewNx = (int(math.ceil((float(remainingapps)/6)*Nx))) - k
278     resizeinstancegroup(ClusterID, 'Task_spot', NewNx)
279     requestedNx.append(str(NewNx))
280     time.sleep(180)
281
282     Timenow = datetime.datetime.now().replace(microsecond=0)
283
284     while (int((Timenow-StartTime).seconds) < int(td*60*60*0.5)):
        #timeelapsed as 0.1 of td in seconds
285         print "WAITING IN LOOP TILL 50% TIME ELAPSED"
286         time.sleep(60)
287         Timenow = datetime.datetime.now().replace(microsecond=0)
288
289     appscompletednow = getclustermetrics(ClusterID, 'AppsCompleted
        ', int((Timenow-StartTime).seconds))
290     remainingapps = 10 - appscompletednow
291     remainingtime = 0.5*td
292
293     print "STAGE 6"
294     NewNx = (int(math.ceil((float(remainingapps)/5)*Nx))) - k
295     resizeinstancegroup(ClusterID, 'Task_spot', NewNx)
296     requestedNx.append(str(NewNx))
```

```
297     time.sleep(180)
298
299     Timenow = datetime.datetime.now().replace(microsecond=0)
300
301     while (int((Timenow-StartTime).seconds) < int(td*60*60*0.6)):
        #timeelapsed as 0.1 of td in seconds
302         print "WAITING IN LOOP TILL 60% TIME ELAPSED"
303         time.sleep(60)
304         Timenow = datetime.datetime.now().replace(microsecond=0)
305
306     appscompletednow = getclustermetrics(ClusterID, 'AppsCompleted
        ', int((Timenow-StartTime).seconds))
307     remainingapps = 10 - appscompletednow
308     remainingtime = 0.4*td
309
310     print "STAGE 7"
311     NewNx = (int(math.ceil((float(remainingapps)/4)*Nx))) - k
312     resizeinstancegroup(ClusterID, 'Task_spot', NewNx)
313     requestedNx.append(str(NewNx))
314     time.sleep(180)
315
316     Timenow = datetime.datetime.now().replace(microsecond=0)
317
318     while (int((Timenow-StartTime).seconds) < int(td*60*60*0.7)):
        #timeelapsed as 0.1 of td in seconds
319         print "WAITING IN LOOP TILL 70% TIME ELAPSED"
320         time.sleep(60)
321         Timenow = datetime.datetime.now().replace(microsecond=0)
322
323     appscompletednow = getclustermetrics(ClusterID, 'AppsCompleted
        ', int((Timenow-StartTime).seconds))
324     remainingapps = 10 - appscompletednow
325     remainingtime = 0.3*td
326
327     print "STAGE 8"
328     NewNx = (int(math.ceil((float(remainingapps)/3)*Nx))) - k
329     resizeinstancegroup(ClusterID, 'Task_spot', NewNx)
330     requestedNx.append(str(NewNx))
331     time.sleep(180)
332
333     Timenow = datetime.datetime.now().replace(microsecond=0)
334
335     while (int((Timenow-StartTime).seconds) < int(td*60*60*0.8)):
        #timeelapsed as 0.1 of td in seconds
336         print "WAITING IN LOOP TILL 80% TIME ELAPSED"
337         time.sleep(60)
338         Timenow = datetime.datetime.now().replace(microsecond=0)
339
340     appscompletednow = getclustermetrics(ClusterID, 'AppsCompleted
        ', int((Timenow-StartTime).seconds))
341     remainingapps = 10 - appscompletednow
342     remainingtime = 0.2*td
343
344     print "STAGE 9"
345     NewNx = (int(math.ceil((float(remainingapps)/2)*Nx))) - k
346     resizeinstancegroup(ClusterID, 'Task_spot', NewNx)
347     requestedNx.append(str(NewNx))
348     time.sleep(180)
349
```

```
350     Timenow = datetime.datetime.now().replace(microsecond=0)
351
352     while (int((Timenow-StartTime).seconds) < int(td*60*60*0.9)):
        #timeelapsed as 0.1 of td in seconds
353         print "WAITING IN LOOP TILL 90% TIME ELAPSED"
354         time.sleep(60)
355         Timenow = datetime.datetime.now().replace(microsecond=0)
356
357     appscompletednow = getclustermetrics(ClusterID, 'AppsCompleted
        ', int((Timenow-StartTime).seconds))
358     remainingapps = 10 - appscompletednow
359     remainingtime = 0.1*td
360
361     print "STAGE 10"
362     NewNx = int(remainingapps*Nx) - k
363     addtasknodes('Task_ondemand', 'ondemand', 'm1.medium', NewNx,
        '0.0', ClusterID)
364     resizeinstancegroup(ClusterID, 'Task_spot', 0)
365     requestedNx.append(str(NewNx))
366     time.sleep(180)
367
368     appscompletednow = getclustermetrics(ClusterID, 'AppsCompleted
        ', int((Timenow-StartTime).seconds))
369     while (appscompletednow < 10):
370         time.sleep(60)
371         appscompletednow = getclustermetrics(ClusterID, '
        AppsCompleted', int((Timenow-StartTime).seconds))
372
373     resizeinstancegroup(ClusterID, 'Task_ondemand', 0)
374     resizeinstancegroup(ClusterID, 'Task_spot', 0)
375
376     print requestedNx
377
378     sys.exit(0)
379     #end of main() definition
380
381 if __name__ == '__main__':
382     main()
```

A.5 Automation Script 2

This script is implementation of **DPB** algorithm.

Listing A.5: Automation of **DPB** Algorithm

```
1  __author__ = 'kabin'
2  #!/usr/bin/python
3
4  import boto3
5  import datetime
6  import paramiko
7  import StringIO
8  import time
9  import math
10 import sys
```

```
from operator import itemgetter


emrclient=boto3.client('emr')
ec2client = boto3.client('ec2')
cwclient = boto3.client('cloudwatch')

#####addtasknode definition#####
def addtasknodes(name, market, insttype, count, spotprice,
    ClusterID):
    #emrclient = boto3.client('emr')
    if market == "ondemand":
        response = emrclient.add_instance_groups(
            InstanceGroups=[
                {
                    'Name': name,
                    'Market': 'ON_DEMAND',
                    'InstanceRole': 'TASK', #'MASTER'|'CORE'|'TASK
    ',
                    'InstanceType': insttype, #e.g. m1.medium
                    'InstanceCount': count,
                }
            ],
            JobFlowId=ClusterID
        )
    elif market == "spot":
        response = emrclient.add_instance_groups(
            InstanceGroups=[
                {
                    'Name': name,
                    'Market': 'SPOT',
                    'InstanceRole': 'TASK', #'MASTER'|'CORE'|'TASK
    ',
                    'BidPrice': spotprice,
                    'InstanceType': insttype, #e.g. m1.medium
                    'InstanceCount': count,
                }
            ],
            JobFlowId=ClusterID
        )
        #end of #addtasknodes(name, market, insttype, count,
    spotprice, ClusterID):
#####addtasknode definition#####
#To add tasknodes do following
#addtasknodes('test1', 'spot', 'm1.medium', 10, '0.02', 'j-10
    TZJU39JUNCC')


#####getspotpricehistory#####
def getspotpricehistoryfrom(productdesc, availabilityzone,
    instancetypes, fromminutes): #return list of max, min, mean,
    median and latest values in duration
    response = ec2client.describe_spot_price_history(
        DryRun = False,
        StartTime = datetime.datetime.utcnow() - datetime.
    timedelta(minutes=fromminutes),
        EndTime = datetime.datetime.utcnow(),
        ProductDescriptions = productdesc, #list e.g. ['Linux/UNIX
    ']
```

```
61          AvailabilityZone = availabilityzone, #e.g. 'us-west-2b'
62          InstanceTypes = instancetypes # list of instance types e.g
        . ['m1.medium']
63      )
64
65      maxprice = max(response['SpotPriceHistory'])['SpotPrice']
66      minprice = min(response['SpotPriceHistory'])['SpotPrice']
67
68      #for mean calculation
69      total = 0.0
70      numdata = len(response['SpotPriceHistory'])
71      for i in range(0, numdata):
72          total += float(response['SpotPriceHistory'][i]['SpotPrice'
        ])
73          #print float(response['SpotPriceHistory'][i]['SpotPrice'])
74      mean = total/numdata
75
76      #for median calculation
77      if numdata%2 ==0:
78          mid1 = numdata/2
79          mid2 = mid1 + 1
80          summid1mid2 = float(sorted(response['SpotPriceHistory'])[
        mid1]['SpotPrice']) + float(sorted(response['SpotPriceHistory'
        ])[mid2]['SpotPrice'])
81          median = summid1mid2/2
82      else:
83          mid = (numdata+1)/2
84          median = sorted(response['SpotPriceHistory'])[mid]['
        SpotPrice']
85
86      #print [float(maxprice), float(minprice), mean, median]
87      return [round(float(maxprice), 4), round(float(minprice), 4),
        round(float(mean), 4), round(float(median), 4), round(float(
        response['SpotPriceHistory'][0]['SpotPrice']), 4)]
88      #end of getspotpricehistory (timefrom, timeto, productdesc,
        availabilityzone, instancetypes)
89  #####getspotpricehistory#####
90
91
92  def getclusterprogress():#returns dictionary with keys as
        Mappercent, reduceperceent, JobCounter, Important EMR Metrics
93      pass
94
95
96  def getclustermetrics(ClusterID, metricsname, timeinseconds):
97      response = cwclient.get_metric_statistics(Namespace = "AWS/
        ElasticMapReduce",
98          MetricName=metricsname,
99          StartTime = datetime.datetime.utcnow() - datetime.
        timedelta(seconds=timeinseconds),
100         EndTime = datetime.datetime.utcnow(),
101         Dimensions=[{'Name': 'JobFlowId', 'Value': ClusterID},],
102         Period = 60,
103         Statistics=['Average']
104     )
105     newlist = sorted(response['Datapoints'], key=itemgetter('
        Timestamp'), reverse=True)
106     if newlist:
107         return int(newlist[0]['Average'])
```

```
108     else:
109         return -1
110 
111 
112 def spotinstancesStatus(ClusterID): #return number of active and
    fulfilled spot instances
113     response = ec2client.describe_spot_instance_requests(
114         DryRun=False,
115         Filters = [
116             {
117               'Name':'availability-zone-group',
118               'Values':[ClusterID],
119             }
120         ]
121     )
122     count = 0
123     for i in range(0, len(response['SpotInstanceRequests'])):
124         if response['SpotInstanceRequests'][i]['State'] == 'active
    ' and response['SpotInstanceRequests'][i]['Status']['Code'] ==
     'fulfilled':
125             count += 1
126     return count
127 
128 
129 def applybidA(count, zone, ClusterID):
130     ondemandprice = 0.087 #for us-west-2 region on-demand m1.
    medium price
131     latestprice = getspotpricehistoryfrom(['Linux/UNIX'], zone, ['
    m1.medium'], 600)[4] #4th index is for latest price
132     bidprice = round(latestprice + (ondemandprice-latestprice)*
    .02, 3) #0.087 is on-demand m1.medium price and bid price
    allowed should be having 3 digits after decimal
133     if bidprice <= 0.6*0.087:
134         addtasknodes('Task_spot', 'spot', 'm1.medium', count, str(
    bidprice), ClusterID)
135 
136 
137 def applybidB(count, zone, ClusterID):
138     ondemandprice = 0.087 #for us-west-2 region on-demand m1.
    medium price
139     latestprice = getspotpricehistoryfrom(['Linux/UNIX'], zone, ['
    m1.medium'], 600)[4] #4th index is for latest price
140     bidprice = round(latestprice + (ondemandprice-latestprice)*
    .05, 3) #0.087 is on-demand m1.medium price and bid price
    allowed should be having 3 digits after decimal
141     if bidprice <= 0.6*0.087:
142         addtasknodes('Task_spot', 'spot', 'm1.medium', count, str(
    bidprice), ClusterID)
143 
144 
145 def checkmastercorestatus(ClusterID):
146     response = emrclient.list_instance_groups(
147         ClusterId=ClusterID
148         )
149     runningcoreinstances = 0
150     runningmasterinstances = 0
151 
152     for i in range(0, len(response['InstanceGroups'])):
153         if response['InstanceGroups'][i]['Name'] == 'Core Instance
```

```
    Group': #this is a list
154             runningcoreinstances = int(response['InstanceGroups'][
    i]['RunningInstanceCount'])
155         if response['InstanceGroups'][i]['Name'] == 'Master
    Instance Group': #this is a list
156             runningmasterinstances = int(response['InstanceGroups'
    ][i]['RunningInstanceCount'])
157
158     if runningmasterinstances == 1 and runningcoreinstances == 2:
    #return True if 1 master and 2 cores are running
159         return True
160     else:
161         return False
162
163
164 def getnumberofrunningnodes(ClusterID, InstanceGroupName):
165     response = emrclient.list_instance_groups(
166         ClusterId = ClusterID
167     )
168
169     runninginstancesingroup = 0
170
171     for i in range(0, len(response['InstanceGroups'])):
172         if response['InstanceGroups'][i]['Name'] ==
    InstanceGroupName:
173             runninginstancesingroup = int(response['InstanceGroups
    '][i]['RunningInstanceCount'])
174
175     return runninginstancesingroup
176
177
178 def resizeinstancegroup(ClusterID, InstanceGroupName, size):
179     response = emrclient.list_instance_groups(
180         ClusterId=ClusterID
181     )
182     #runninginstancesingroup = 0
183     IdList = []
184
185     for i in range(0, len(response['InstanceGroups'])):
186         #print response['InstanceGroups'][i]
187         if response['InstanceGroups'][i]['Name'] ==
    InstanceGroupName: #this is a list
188             #runninginstancesingroup = int(response['
    InstanceGroups'][i]['RunningInstanceCount'])
189             IdList.append(response['InstanceGroups'][i]['Id'])
190
191     for Ids in IdList:
192         response1 = emrclient.modify_instance_groups(
193                     InstanceGroups = [
194                         {
195                             'InstanceGroupId': Ids,
196                             'InstanceCount' : size
197                         }
198                     ]
199                 )
200
201
202 def main():
203
```

```
204     StartTime = datetime.datetime.now().replace(microsecond=0)
205     #requestedNx = []
206     ondemandstatus = 0
207
208     #INITIAL VALUES AFTER CLUSTER IS RUN AND ESTIMATION IS DONE
209     td = 10 #deadline is 10 hours
210     k = 2 #number of core instancees
211     Nx = 6 #get from estimation
212     ClusterID = 'j-ZWX9OQJY49R9' #get after cluster is started
213     availabilityzone = 'us-west-2c' #get after cluster is run in
        us-west-2
214
215     appscompletednow = 0
216
217     #At 0% time elapsed - STAGE 1
218     print "STAGE 1"
219     applybidA(int(2*Nx-k), availabilityzone, ClusterID) #bid price
         is median of 10 hours price + G
220     time.sleep(180)
221
222     Timenow = datetime.datetime.now().replace(microsecond=0)
223     while (int((Timenow-StartTime).seconds) < int(td*60*60*0.1)):
        #timeelapsed as 0.1 of td in seconds
224         print "WAITING IN LOOP TILL 10% TIME ELAPSED"
225         time.sleep(60)
226         Timenow = datetime.datetime.now().replace(microsecond=0)
227
228     if (int(spotinstancesStatus(ClusterID)) == int(2*Nx-k)):
229         pass
230     else:
231         resizeinstancegroup(ClusterID, "Task_spot", 0)
232         time.sleep(120)
233         applybidA(int(2*Nx-k), availabilityzone, ClusterID) #bid
        price is median of 10 hours price + G
234
235
236     print "STAGE 2"
237     Timenow = datetime.datetime.now().replace(microsecond=0)
238     while (int((Timenow-StartTime).seconds) < int(td*60*60*0.2)):
        #timeelapsed as 0.1 of td in seconds
239         print "WAITING IN LOOP TILL 20% TIME ELAPSED"
240         time.sleep(60)
241         Timenow = datetime.datetime.now().replace(microsecond=0)
242
243     if (int(spotinstancesStatus(ClusterID)) == int(2*Nx-k)):
244         pass
245     else:
246         resizeinstancegroup(ClusterID, "Task_spot", 0)
247         time.sleep(120)
248         applybidA(int(2*Nx-k), availabilityzone, ClusterID) #bid
        price is median of 10 hours price + G
249
250     print "STAGE 3"
251     Timenow = datetime.datetime.now().replace(microsecond=0)
252     while (int((Timenow-StartTime).seconds) < int(td*60*60*0.3)):
        #timeelapsed as 0.1 of td in seconds
253         print "WAITING IN LOOP TILL 30% TIME ELAPSED"
254         time.sleep(60)
255         Timenow = datetime.datetime.now().replace(microsecond=0)
```

```
256
257     if (int(spotinstancesStatus(ClusterID)) == int(2*Nx-k)):
258         pass
259     else:
260         resizeinstancegroup(ClusterID, "Task_spot", 0)
261         time.sleep(120)
262         applybidA(int(2*Nx-k), availabilityzone, ClusterID) #bid
    price is median of 10 hours price + G
263
264
265     print "STAGE 4"
266     Timenow = datetime.datetime.now().replace(microsecond=0)
267     while (int((Timenow-StartTime).seconds) < int(td*60*60*0.4)):
    #timeelapsed as 0.1 of td in seconds
268         print "WAITING IN LOOP TILL 40% TIME ELAPSED"
269         time.sleep(60)
270         Timenow = datetime.datetime.now().replace(microsecond=0)
271
272     if (int(spotinstancesStatus(ClusterID)) == int(2*Nx-k)):
273         pass
274     else:
275         resizeinstancegroup(ClusterID, "Task_spot", 0)
276         time.sleep(120)
277         applybidA(int(2*Nx-k), availabilityzone, ClusterID) #bid
    price is median of 10 hours price + G
278
279
280     print "STAGE 5"
281
282     Timenow = datetime.datetime.now().replace(microsecond=0)
283     while (int((Timenow-StartTime).seconds) < int(td*60*60*0.5)):
    #timeelapsed as 0.1 of td in seconds
284         print "WAITING IN LOOP TILL 50% TIME ELAPSED"
285         time.sleep(60)
286         Timenow = datetime.datetime.now().replace(microsecond=0)
287
288
289     print "Checkpoint 1"
290
291     appscompletednow = getclustermetrics(ClusterID, 'AppsCompleted
    ', int((Timenow-StartTime).seconds))
292
293     if appscompletednow <=4:
294         resizeinstancegroup(ClusterID, "Task_spot", 0)
295         addtasknodes('Task_ondemand', 'ondemand', 'm1.medium', Nx,
     '0.0', ClusterID)
296         time.sleep(120)
297         applybidB(int(2*Nx-k), availabilityzone, ClusterID)
298         ondemandstatus = 1
299
300     elif appscompletednow ==5:
301         resizeinstancegroup(ClusterID, "Task_spot", 0)
302         time.sleep(120)
303         applybidB(int(2*Nx-k), availabilityzone, ClusterID)
304
305     elif appscompletednow >5 and appscompletednow<=9:
306         if (int(spotinstancesStatus(ClusterID)) == int(2*Nx-k)):
307             pass
308         else:
```

```
309             resizeinstancegroup(ClusterID, "Task_spot", 0)
310             time.sleep(120)
311             applybidA(int(2*Nx-k), availabilityzone, ClusterID)
312
313     elif appscompletednow == 10:
314         resizeinstancegroup(ClusterID, 'Task_spot', 0)
315         if ondemandstatus == 1:
316             resizeinstancegroup(ClusterID, 'Task_ondemand', 0)
317         sys.exit(0)
318
319
320     print "Checkpoint 2"
321     Timenow = datetime.datetime.now().replace(microsecond=0)
322     while (int((Timenow-StartTime).seconds) < int(td*60*60*0.8)):
    #timeelapsed as 0.1 of td in seconds
323         print "WAITING IN LOOP TILL 60%-80% TIME ELAPSED"
324         time.sleep(60)
325         Timenow = datetime.datetime.now().replace(microsecond=0)
326         appscompletednow = getclustermetrics(ClusterID, '
    AppsCompleted', int((Timenow-StartTime).seconds))
327         if appscompletednow == 10:
328             if ondemandstatus ==1:
329                 resizeinstancegroup(ClusterID, 'Task_ondemand', 0)
330             resizeinstancegroup(ClusterID, 'Task_spot', 0)
331             sys.exit(0)
332
333
334     appscompletednow = getclustermetrics(ClusterID, 'AppsCompleted
    ', int((Timenow-StartTime).seconds))
335
336     if appscompletednow <= 7:
337         NewNx = int(math.ceil((float(10-appscompletednow)/2)*Nx))
    - k
338         if ondemandstatus == 1:
339             resizeinstancegroup(ClusterID, 'Task_ondemand', NewNx)
340             resizeinstancegroup(ClusterID, 'Task_spot', 0)
341             time.sleep(120)
342             applybidB(Nx, availabilityzone, ClusterID)
343         elif ondemandstatus ==0:
344             addtasknodes('Task_ondemand', 'ondemand', 'm1.medium',
     NewNx, '0.0', ClusterID)
345             resizeinstancegroup(ClusterID, 'Task_spot', 0)
346             time.sleep(120)
347             applybidB(Nx, availabilityzone, ClusterID)
348
349     elif appscompletednow >7 and appscompletednow <10:
350         if ondemandstatus == 1:
351             resizeinstancegroup(ClusterID, 'Task_ondemand', Nx-k)
352             resizeinstancegroup(ClusterID, 'Task_spot', 0)
353             time.sleep(120)
354             applybidB(Nx, availabilityzone, ClusterID)
355         elif ondemandstatus == 0:
356             addtasknodes('Task_ondemand', 'ondemand', 'm1.medium',
     Nx-k, '0.0', ClusterID)
357             resizeinstancegroup(ClusterID, 'Task_spot', 0)
358             time.sleep(120)
359             applybidB(Nx, availabilityzone, ClusterID)
360
361     Timenow = datetime.datetime.now().replace(microsecond=0)
```

```
    while (int((Timenow-StartTime).seconds) < int(td*60*60)): #
    timeelapsed as 0.1 of td in seconds
        print "WAITING IN LOOP TILL 80%-100% TIME ELAPSED"
        time.sleep(60)
        Timenow = datetime.datetime.now().replace(microsecond=0)
        appscompletednow = getclustermetrics(ClusterID, '
    AppsCompleted', int((Timenow-StartTime).seconds))
        if appscompletednow == 10:
            if ondemandstatus ==1:
                resizeinstancegroup(ClusterID, 'Task_ondemand', 0)
            resizeinstancegroup(ClusterID, 'Task_spot', 0)
            sys.exit(0)

    sys.exit(0)
    #end of main() definition

if __name__ == '__main__':
    main()
```

A.6 Python Script to provision/run EMR cluster and adding steps

The following codes provision and run Amazon EMR cluster and also adds ten steps that points to equally divided files in ten folders for processing.

Listing A.6: Provisioning/running EMR cluster with Hadoop and adding steps

```
__author__ = 'kabin'
#!/usr/bin/python

import boto3
from datetime import datetime

emrclient = boto3.client('emr')
response = emrclient.run_job_flow(Name="mycluster01",
  LogUri='s3://aws-test-kabin01/logs',
  ReleaseLabel="emr-4.6.0",
  Instances={'MasterInstanceType':'m1.medium',
  'SlaveInstanceType':'m1.medium',
  'InstanceCount':3,
  'Ec2KeyName':'EMR_test',
        'KeepJobFlowAliveWhenNoSteps':True
        },
        Configurations=[{"Classification": "emrfs-site",
          "Properties": {"fs.s3.consistent": "true"}}
          ],
        JobFlowRole="EMR_EC2_DefaultRole",
        ServiceRole="EMR_DefaultRole"
        )

ClusterID = response['JobFlowId']

responseaddsteps = emrclient.add_job_flow_steps(
```

```
JobFlowId=ClusterID,
Steps=[
{
  'Name': 'Step01',
  'ActionOnFailure':'CONTINUE',
  'HadoopJarStep': {
  'Jar': 's3://aws-test-kabin01/script/log-analysis.jar',
              'MainClass': 'None',
  'Args': [
    'LogAnalysisDriver',
    's3n://aws-test-kabin01/data/01/*',
    's3n://aws-test-kabin01/run01',
    ]
  }
},
{
  'Name': 'Step02',
  'ActionOnFailure':'CONTINUE',
  'HadoopJarStep': {
  'Jar': 's3://aws-test-kabin01/script/log-analysis.jar',
              'MainClass': 'None',
  'Args': [
    'LogAnalysisDriver',
    's3n://aws-test-kabin01/data/02/*',
    's3n://aws-test-kabin01/run02',
    ]
  }
},
{
  'Name': 'Step03',
  'ActionOnFailure':'CONTINUE',
  'HadoopJarStep': {
  'Jar': 's3://aws-test-kabin01/script/log-analysis.jar',
              'MainClass': 'None',
  'Args': [
    'LogAnalysisDriver',
    's3n://aws-test-kabin01/data/03/*',
    's3n://aws-test-kabin01/run03',
    ]
  }
},
{
  'Name': 'Step04',
  'ActionOnFailure':'CONTINUE',
  'HadoopJarStep': {
  'Jar': 's3://aws-test-kabin01/script/log-analysis.jar',
              'MainClass': 'None',
  'Args': [
    'LogAnalysisDriver',
    's3n://aws-test-kabin01/data/04/*',
    's3n://aws-test-kabin01/run04',
    ]
  }
},
{
  'Name': 'Step05',
  'ActionOnFailure':'CONTINUE',
  'HadoopJarStep': {
  'Jar': 's3://aws-test-kabin01/script/log-analysis.jar',
```

```
                'MainClass': 'None',
    'Args': [
      'LogAnalysisDriver',
      's3n://aws-test-kabin01/data/05/*',
      's3n://aws-test-kabin01/run05',
      ]
    }
  },
  {
    'Name': 'Step06',
    'ActionOnFailure':'CONTINUE',
    'HadoopJarStep': {
    'Jar': 's3://aws-test-kabin01/script/log-analysis.jar',
                'MainClass': 'None',
    'Args': [
      'LogAnalysisDriver',
      's3n://aws-test-kabin01/data/06/*',
      's3n://aws-test-kabin01/run06',
      ]
    }
  },
  {
    'Name': 'Step07',
    'ActionOnFailure':'CONTINUE',
    'HadoopJarStep': {
    'Jar': 's3://aws-test-kabin01/script/log-analysis.jar',
                'MainClass': 'None',
    'Args': [
      'LogAnalysisDriver',
      's3n://aws-test-kabin01/data/07/*',
      's3n://aws-test-kabin01/run07',
      ]
    }
  },
  {
    'Name': 'Step08',
    'ActionOnFailure':'CONTINUE',
    'HadoopJarStep': {
    'Jar': 's3://aws-test-kabin01/script/log-analysis.jar',
                'MainClass': 'None',
    'Args': [
      'LogAnalysisDriver',
      's3n://aws-test-kabin01/data/08/*',
      's3n://aws-test-kabin01/run08',
      ]
    }
  },
  {
    'Name': 'Step09',
    'ActionOnFailure':'CONTINUE',
    'HadoopJarStep': {
    'Jar': 's3://aws-test-kabin01/script/log-analysis.jar',
                'MainClass': 'None',
    'Args': [
      'LogAnalysisDriver',
      's3n://aws-test-kabin01/data/09/*',
      's3n://aws-test-kabin01/run09',
      ]
    }
```

```
    },
    {
      'Name': 'Step10',
      'ActionOnFailure':'CONTINUE',
      'HadoopJarStep': {
      'Jar': 's3://aws-test-kabin01/script/log-analysis.jar',
                  'MainClass': 'None',
      'Args': [
        'LogAnalysisDriver',
        's3n://aws-test-kabin01/data/10/*',
        's3n://aws-test-kabin01/run10',
        ]
      }
    }
    ]
)
```

A.7 AWS CLI Command to provision/run EMR cluster and adding steps

The following AWS Command Line Interface command provision and run Amazon EMR cluster and and also adds ten steps that points to equally divided files in ten folders for processing.

Listing A.7: Provisioning/running EMR cluster with Hadoop and adding steps

```
aws emr create-cluster --applications Name=Hadoop --ec2-attributes
'{"KeyName":"EMR_test","InstanceProfile":"EMR_EC2_DefaultRole",
"SubnetId":"subnet-5e736c3b",
"EmrManagedSlaveSecurityGroup":"sg-66bfce01",
"EmrManagedMasterSecurityGroup":"sg-67bfce00"}'
--service-role EMR_DefaultRole --enable-debugging
--release-label emr-4.6.0
--log-uri 's3n://aws-test-kabin01/logs/'
--steps '[{"Args":["LogAnalysisDriver","s3n://aws-test-kabin01/
    data03/10/*",
"s3n://aws-test-kabin01/runcluster2c/run10"],
"Type":"CUSTOM_JAR","ActionOnFailure":"CONTINUE",
"Jar":"s3://aws-test-kabin01/script/log-analysis.jar",
"Properties":"","Name":"Step10"},
{"Args":["LogAnalysisDriver","s3n://aws-test-kabin01/data03/09/*",
"s3n://aws-test-kabin01/runcluster2c/run9"],
"Type":"CUSTOM_JAR","ActionOnFailure":"CONTINUE",
"Jar":"s3://aws-test-kabin01/script/log-analysis.jar",
"Properties":"","Name":"Step09"},
{"Args":["LogAnalysisDriver","s3n://aws-test-kabin01/data03/08/*",
"s3n://aws-test-kabin01/runcluster2c/run8"],
"Type":"CUSTOM_JAR","ActionOnFailure":"CONTINUE",
"Jar":"s3://aws-test-kabin01/script/log-analysis.jar",
"Properties":"","Name":"Step08"},
{"Args":["LogAnalysisDriver","s3n://aws-test-kabin01/data03/07/*",
"s3n://aws-test-kabin01/runcluster2c/run7"],
"Type":"CUSTOM_JAR","ActionOnFailure":"CONTINUE",
```

```
"Jar":"s3://aws-test-kabin01/script/log-analysis.jar",
"Properties":"","Name":"Step07"},
{"Args":["LogAnalysisDriver","s3n://aws-test-kabin01/data03/06/*",
"s3n://aws-test-kabin01/runcluster2c/run6"],
"Type":"CUSTOM_JAR","ActionOnFailure":"CONTINUE",
"Jar":"s3://aws-test-kabin01/script/log-analysis.jar",
"Properties":"","Name":"Step06"},
{"Args":["LogAnalysisDriver","s3n://aws-test-kabin01/data03/05/*",
"s3n://aws-test-kabin01/runcluster2c/run5"],
"Type":"CUSTOM_JAR","ActionOnFailure":"CONTINUE",
"Jar":"s3://aws-test-kabin01/script/log-analysis.jar",
"Properties":"","Name":"Step05"},
{"Args":["LogAnalysisDriver","s3n://aws-test-kabin01/data03/04/*",
"s3n://aws-test-kabin01/runcluster2c/run4"],
"Type":"CUSTOM_JAR","ActionOnFailure":"CONTINUE",
"Jar":"s3://aws-test-kabin01/script/log-analysis.jar",
"Properties":"","Name":"Step04"},
{"Args":["LogAnalysisDriver","s3n://aws-test-kabin01/data03/03/*",
"s3n://aws-test-kabin01/runcluster2c/run3"],
"Type":"CUSTOM_JAR","ActionOnFailure":"CONTINUE",
"Jar":"s3://aws-test-kabin01/script/log-analysis.jar",
"Properties":"","Name":"Step03"},
{"Args":["LogAnalysisDriver","s3n://aws-test-kabin01/data03/02/*",
"s3n://aws-test-kabin01/runcluster2c/run2"],
"Type":"CUSTOM_JAR","ActionOnFailure":"CONTINUE",
"Jar":"s3://aws-test-kabin01/script/log-analysis.jar",
"Properties":"","Name":"Step02"},
{"Args":["LogAnalysisDriver","s3n://aws-test-kabin01/data03/01/*",
"s3n://aws-test-kabin01/runcluster2c/run1"],
"Type":"CUSTOM_JAR","ActionOnFailure":"CONTINUE",
"Jar":"s3://aws-test-kabin01/script/log-analysis.jar",
"Properties":"","Name":"Step01"}]'
--name 'Cluster_2c' --instance-groups '[{"InstanceCount":2,
"InstanceGroupType":"CORE",
"InstanceType":"m1.medium","Name":"Core Instance Group"},
{"InstanceCount":1,"InstanceGroupType":"MASTER",
"InstanceType":"m1.medium","Name":"Master Instance Group"}]'
--region us-west-2
```