PROJECT NO. 50

NKR: ON TOP SCHEDULER FOR APACHE MESOS

MS.PASINEE SANTIVORRANANT
MR.SUPAPAT SRI-ON
MS.PARATTHA WEERAPONG

A PROJECT SUBMITTED IN PARTIAL FULFILLMENT
OF THE REQUIREMENTS FOR
THE DEGREE OF BACHELOR OF ENGINEERING (COMPUTER ENGINEERING)
FACULTY OF ENGINEERING
KING MONGKUT'S UNIVERSITY OF TECHNOLOGY THONBURI
2020

Project No. 50

NKR: On top scheduler for Apache Mesos

Ms.Pasinee Santivorranant
Mr.Supapat Sri-on
Ms.Parattha Weerapong

A Project Submitted in Partial Fulfillment
of the Requirements for
the Degree of Bachelor of Engineering (Computer Engineering)
Faculty of Engineering
King Mongkut's University of Technology Thonburi
2020

Project Committee

.... (Asst Prof.Rajchawit Sarochawikasit)	Project Advisor
.... (Asst Prof. Dr. Khajonpong Akkarajitsakul)	Committee Member
.... (Asst Prof. Dr. Phond Phunchongharn)	Committee Member
.... (Asst Prof. Sanan Srakaew)	Committee Member

Project Title	Project No. 50
	NKR: On top scheduler for Apache Mesos
Credits	3
Member(s)	Ms.Pasinee Santivorranant
	Mr.Supapat Sri-on
	Ms.Parattha Weerapong
Project Advisor	Asst Prof.Rajchawit Sarochawikasit
Program	Bachelor of Engineering
Field of Study	Computer Engineering
Department	Computer Engineering
Faculty	Engineering
Academic Year	2020

Abstract

Keywords: Apache Mesos / Scheduling / Dominant Resource Fairness / Multi-tenant / Fault tolerant / Artificial Intelligence

หัวข้อปริญญานิพนธ์	หัวข้อปริญญานิพนธ์บรรทัดแรก หัวข้อปริญญานิพนธ์บรรทัดสอง
หน่วยกิต	3
ผู้เขียน	นางสาวภาสินี สันติวรนันท์ นายศุภพัฒน์ ศรีอ่อน นางสาวปรัษฐา วีระพงษ์
อาจารย์ที่ปรึกษา	ผศ.ดร.ราชวิชช์ สโรชวิกสิต
หลักสูตร	วิศวกรรมศาสตรบัณฑิต
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
ภาควิชา	วิศวกรรมคอมพิวเตอร์
คณะ	วิศวกรรมศาสตร์
ปีการศึกษา	2563

บทคัดย่อ

คำสำคัญ: Apache Mesos / Scheduling / Dominant Resource Fairness / Multi-tenant / Fault tolerant / Artificial Intelligence

ACKNOWLEDGMENTS

Acknowledge your advisors and thanks your friends here..

CONTENTS

PAGE

LIST OF TABLES

TABLE **PAGE**

LIST OF FIGURES

FIGURE **PAGE**

LIST OF SYMBOLS

SYMBOL		UNIT
α	Test variable	m^2
λ	Interarival rate	jobs/ second
μ	Service rate	jobs/ second

LIST OF TECHNICAL VOCABULARY AND ABBREVATIONS

ABC	=	Adaptive Bandwidth Control
MANET	=	Mobile Ad Hoc Network

CHAPTER 1 INTRODUCTION

1.1 Problem Statement and Approach

Nowadays, several different types of applications, which are short or long-lived jobs, container orchestration, or MPI jobs, are executed in clouds or large computer clusters. Multiple users can demand difference resources to execute their tasks. Apache Mesos is a Middleware for the data center by introducing an abstraction layer that provides an entire data centers as a single large server. Instead of focusing on one application that running on a specific server. Mesos resource-isolation allows multi-tenant — the ability to run multiple applications on a single machine. Default sharing for multiple resources in this multi-tenant environment is defined by the Dominant Resource Fairness (DRF). Mesos receives the resources based on their current usage, which are responsible for scheduling their tasks within the allocation. In multiple schedulers can cause the fairness-imbalance in a multi-user environment, liked a greedy scheduler. It consumes more than its share of resources. Running multiple small tasks is better than launching large ones in terms of time spent waiting for enough resources.

Therefore, this project aims to improve the fairness of the scheduler by reducing the unfair waiting time due to higher resource demand in a pending task list and use log data to improve the whole cluster.

1.2 Objectives

- To study about job scheduling in Apache Mesos
- To study how to develop an algorithm to improve performance of scheduler in large-scale clustered environments.
- · To evaluate result and compare with Apache Mesos scheduler by using difference job types in the list (short job, long job, MPI)

1.3 Scope

- This project focuses on the reduction of job failed.
- Design and develop an add-on architecture on top of the Apache Mesos scheduler, to track and distribute the incoming tasks.
- What are the limitations of existing approaches?

1.4 Tasks and Schedule

Table 1.1 Semester 1’s Gantt chart

Task/Week	August				September				October				November			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
1.Idea Document																
1.1 Find interesting problems																
1.2 Brainstorm ideas and choose topic																
1.3 Project discussion with advisor																
1.4 Write idea document report																
2.Proposal																
2.1 Explore related work and technologies																
2.2 Task breakdown																
2.3 Gantt chart																
2.4 Write proposal																
2.5 Present proposal																
3.Semester Report																
3.1 Literature review																
3.2 Design architecture diagram																
3.3 Design sequence diagram																
3.4 Write semester diagram																
3.5 Present Semester report																
4.Setup project & preparation																
4.1 Setup cluster & framework application																
4.2 Observe sharing and waiting time in queue for each framework																
4.3 Gathering server logs																

Table 1.2 Semester 2's Gantt chart

Task/Week	January				February				March				April				May			
	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4	1	2	3	4
5.Implementation																				
5.1 Implement new on top scheduler for the whole cluster scheduling																				
5.2 Train model for scheduler prediction																				
6.Evaluation																				
6.1 Evaluate new on top scheduler with many situations																				
6.2 Evaluate model for scheduler prediction																				
6.3 Improve way to distributed framework																				
6.4 Tune model for scheduler prediction																				
7.Final Report																				
7.1 Write final report																				
7.2 Present final report																				

CHAPTER 2 BACKGROUND KNOWLEDGE AND LITERATURE REVIEW

2.1 Knowledge Background

In 2009, Apache Mesos [6] was a research project at the University of California in Berkeley. Benjamin, et al. wanted to improve datacenter efficiency by allowing multiple applications to share a single computing cluster across the many servers that make up a modern datacenter. So, multiple applications can share the processor, memory, and hard drive with any laptop or workstation. In 2010, the Mesos project entered the Apache Incubator, an arm of the Apache Software Foundation, so this project can gain the full support of the ASF's efforts. In 2013, The Apache Mesos project graduated from the incubator and founded Mesosphere. Mesosphere's flagship product, the Datacenter Operating System (DCOS), commercializes the open-source project by providing a turnkey solution to enterprises looking to deploy applications and scale infrastructure as effortlessly as other companies using Mesos, such as Airbnb, Apple, and Netflix.

Meanwhile, most such operating systems only fairly divide and account for CPU cycles. So, performance isolation is essential to operating systems shared by dependable services. These dependable services require specifying and enforcing policies for all resources, and that current metrics for evaluating fair sharing are insufficient. In 2006, Aage Kvalnes et al. researched new policy specifications and metrics, and illustrated these with the help of a new operating system that supports holistic resource sharing. [17]

In data centers and clouds, where applications could be co-scheduled on the same physical nodes, resource fairness needs to extend to multiple resource types such as memory, disk I/O, and network bandwidth. Ali, et al. considered the problem of fair resource allocation in a system containing different resource types, where each user may have different demands for each resource and researched about a new generalization of max-min fairness to multiple resource types called Dominant Resource Fairness (DRF). [5]

2.2 Theoretical and Core Concepts

2.2.1 Tasks Failures Detection

Hadoop usually uses JobTracker to detect failures of the TaskTracker nodes. It detects with heartbeat-based failure detection. The TaskTracker will send heartbeat message to JobTracker and JobTracker will declare a TaskTracker as dead only when it does not receive heartbeat for a limited time. It cannot quickly detect the failures and it may assign task to dead nodes. This can increase the number of failure tasks in Hadoop. [15]

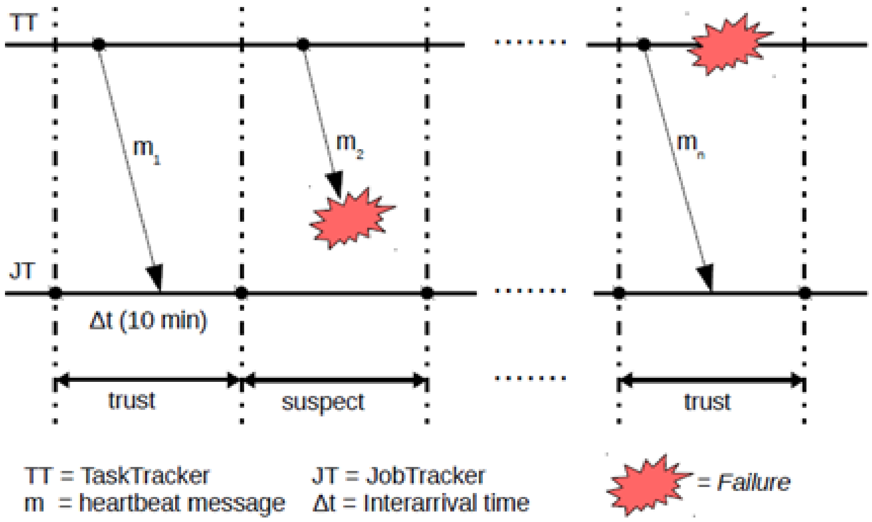


Figure 2.1 TaskTracker Failure Detection Model in Hadoop Framework

[From: Adaptive Failure-Aware Scheduling for Hadoop. (Mbarka Soualhia, Montreal, Quebec, Canada, 2018)]

For example, active TaskTracker send heartbeat messages to JobTracker every 3 seconds. While JobTracker check the timeout condition every 200 seconds. And there are network delays or messages losses, so some heartbeat may arrive late or loss. The JobTracker may consider that TaskTracker as dead node even it is available as shown in **Figure** 2.1. that heartbeat message m2 does not arrive and the JobTracker consider this TaskTracker as dead.

2.2.2 Container Technology

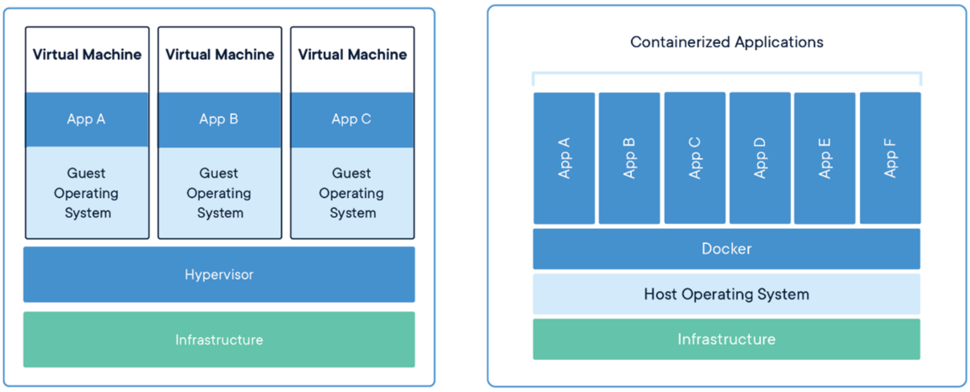


Figure 2.2 Virtual machine vs Container

[From: Docker, "What is a Container?," [Online]. Available: www.docker.com/resources/what-container. Accessed 30 August 2020]

Container Technology is a method to package up code and all its dependencies, so the application runs quickly and reliably from one computing environment to another, with container software having names including the popular choices of Docker, Apache Mesos, RKT, and Kubernetes. The virtual machine contains the entire operating system. Therefore, the physical server that runs several virtual machines is running several operating systems' simultaneously as shown in **Figure** 2.2. [2]

There is a lot of overhead on virtual machine. In contrast, with container technology, the server runs a single operating system. Each container can share this single operating system with other containers on server. Containers require less resource of server with less overhead and more efficient than virtual machines. [1] Containers are set up to accomplish work in a multiple container architecture (container cluster). They also enable a program to be broken down into smaller pieces, which are known as microservices. So, the program can work on each of the containers separately.

2.2.3 Overview of machine learning

Machine learning is a branch of artificial intelligence (AI). It is the machine's ability to learn from data provided without human intervention and able to improve decision from experience. Without human directed programming instructions, the machine accesses data, observes and finds data pattern. The more data input the better data pattern learning and better decision making.[3]

2.2.4 AI - Artificial Neural network

An artificial neural network (ANN) is a computational model imitates the natural human brain. The network consists of hundreds or thousands small neuron nodes. Those numerous neuron nodes communicate to each other in the web form. The neuron node is called processing unit. Each of processing unit is interconnected by nodes. Each processing unit comprises of input unit and output unit. Input unit receives various type of data format. It also has an internal weighting system. The neural network learns from the input and produce output result.

ANN use rules and guidelines to generate result/ output. The set of these learning rules is called backpropagation because it uses backward propagation of error to learn or improve the better result. ANN learns data patterns in training phase. It compares actual output with the desired output that is expected result in supervised phase. The difference between actual and expected result are worked backward to adjust the weight of its connections between the units. The purpose is to make the lowest possible error. [4]

2.2.5 Deep learning

Nowadays, there are collections of vast unlabeled and unstructured data gathering from various sources that is difficult to analyses useful information by traditional programs in a linear way. The hierarchical level of artificial neural network that work in web form to process data with a nonlinear approach is called Deep Learning. The first layer of the neural network processes a raw data and pass output on to the next layer. The second layer processes first layer output plus additional information and pass output to next layer again. This continues across all levels of the neuron network to make information more meaningful information. [12]

Deep learning models can achieve high accuracy, sometimes exceeding human-level performance. "Deep" refers to the powerful number of hidden layers in the neural network. However, it needs lots of labeled data and high-performance machines to analyze. The organized layers of interconnected nodes can be tens or hundreds of hidden layers. [16] as shown in **Figure** 2.3.

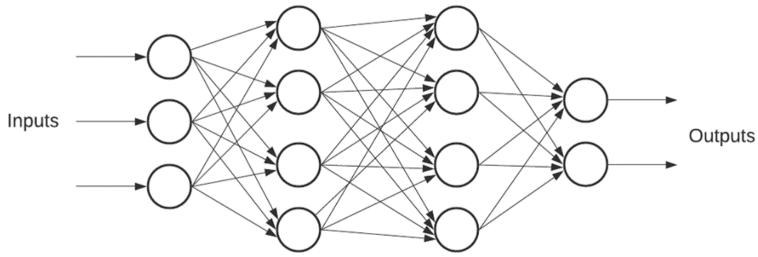


Figure 2.3 Neural networks.

2.3 Technologies survey

2.3.1 Apache Mesos

Mesos consists of a master, agent daemons running on each cluster node, and Mesos frameworks that run task on these agents as shown in **Figure** 2.4. Architecture consist of three components: masters, slaves, and the frameworks that run on them. Mesos relies on Apache ZooKeeper, a distributed database used specifically for coordinating leader election within the cluster, and for leader detection by other Mesos masters, slaves, and frameworks. [8]

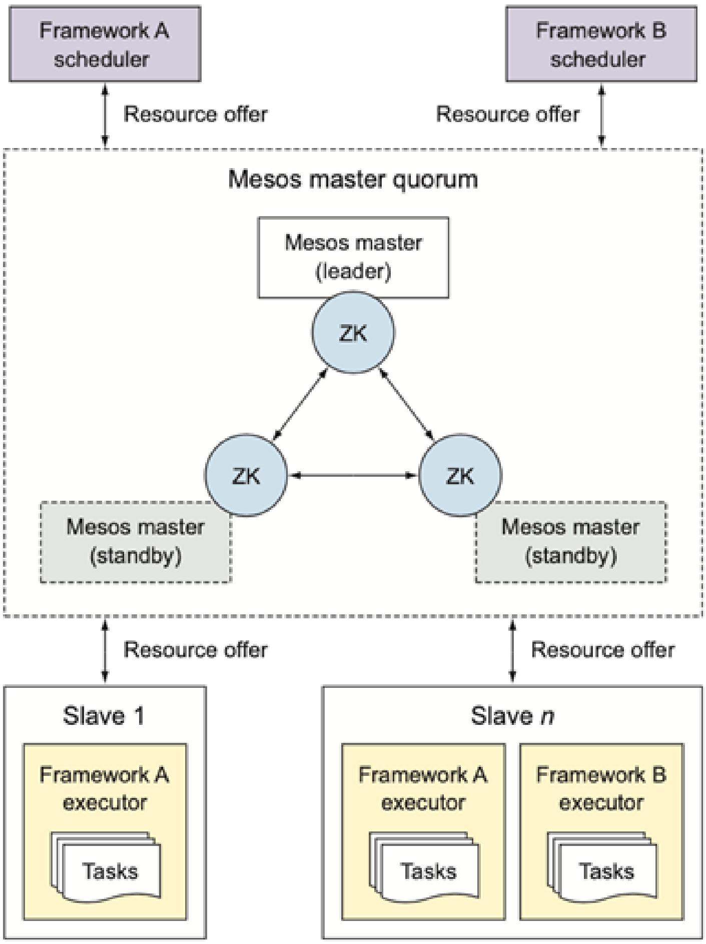


Figure 2.4 The Mesos architecture consists of one or more masters, slaves, and frameworks.

[From: Ignazio, R. Mesos in Action (Manning Publications Co., Shelter Island, NY, 2016).]

1. **Masters:** Mesos masters are responsible for managing the Mesos slave daemons running on each machine in the cluster. Using Zookeeper, they coordinate which node will be the leading master, and which masters will be on standby, and ready to take over if the leading master goes offline. A Mesos cluster requires minimum one master, and three or more are recommended for production deployments. Zookeeper can run on the same machines as the Mesos masters themselves or use a standalone Zookeeper cluster.
2. **Slaves:** The machines in a cluster responsible for executing a framework's tasks.

3. **Frameworks:** Mesos application that's responsible for scheduling and executing tasks on a cluster. A framework is made up of two components: a scheduler and an executor.

    - **Schedulers** A scheduler is a long-running service responsible for connecting to a Mesos master and accepting or rejecting resource offers. Mesos delegates the responsibility of scheduling to the framework, instead of attempting to schedule all the work for a cluster itself. The scheduler can then accept or reject a resource offer based on whether it has any tasks to run at the time of the offer.
    - **Executor** An executor is a process launched on a Mesos slave that runs a framework's tasks on a slave.

Dominant resource is a resource of specific type (CPU, memory, disk, ports) which is most demanded by given framework among other resources it needs. DRF computes the share of resource allocated to a framework (dominant share) and tries to maximize the smallest dominant share in the system. for next round offers the resources first to the one with smallest dominant share, then to the second smallest one and so on. [9] Example with 9 CPUs and 18 GB RAM to two frameworks running task that require <1 CPU, 4GB> and <3CPUs, 1GB> shown in **Table** 2.1.

Table 2.1 Example of running 2 frameworks.

Schedule	Framework A		Framework B		total	
	Resource Share	Dominant Share	Resource Share	Dominant Share	CPU	RAM
B	<0, 0>	0	<3/9, 1/18>	1/3	3/9	1/18
A	<1/9>, <4/18>	**2/9**	<3/9, 1/18>	1/3	4/9	5/18
A	<2/9>, <8/18>	4/9	<3/9>, <1/18>	**1/3**	5/9	8/18
B	<2/9>, <8/18>	**4/9**	<6/9>, 2/18>	2/3	8/9	10/18
A	<3/9>, <12/18>	2/3	<6/9>, 2/18>	2/3	1	14/18

2.3.2 Zookeeper

ZooKeeper is an open-source Apache project that provides a centralized service for providing configuration information, naming, synchronization, and group services over large clusters in distributed systems. The goal is to make these systems easier to manage with improved, more reliable propagation of changes. ZooKeeper provides an infrastructure for cross-node synchronization by maintaining status type information in memory on ZooKeeper servers. A ZooKeeper server keeps a copy of the state of the entire system and persists this information in local log files. Large Hadoop clusters are supported by multiple ZooKeeper servers, with a master server synchronizing the top-level servers. [7]

2.3.3 Elasticsearch

Elasticsearch is an open-source search and analytics engine built on Apache Lucene and developed in Java. Elasticsearch can be used to store and search all kinds of documents, analyze huge volumes of data in near real-time and give back answers in milliseconds, and supports multitenancy. It's able to achieve fast search responses because it searches an index directly instead of searching the text. Related data is often stored in the same index, which consists of one or more primary shards, and zero or more replica shards. Once an index has been created, the number of primary shards cannot be changed. It uses a structure based on documents and comes with extensive REST APIs for storing and searching the data.

Elasticsearch is the component of the Elastic Stack, a set of open-source tools for storage, analysis, and visualization. The four components, Elasticsearch, Logstash, Kibana and Beats, are designed for use as an integrated solution. It is commonly referred to as the "ELK" stack.[13]

2.3.4 Chronos

Chronos is the Mesos Cron system. It handles time-based scheduling of jobs on a Mesos cluster. Chronos can be used to schedule commands or scripts. The Chronos feature set is easily and reliably to create standalone schedule-based jobs, as well as complex dependency-based jobs and pipelines, simply by specifying the schedule and resources that the job requires. This guarantees that time-based jobs are running on time while continuing to use datacenter resources as efficiently as possible. In **Figure** 2.5, it shows about the differences between running Cron jobs on a single machine and running them on a Mesos cluster.[8]

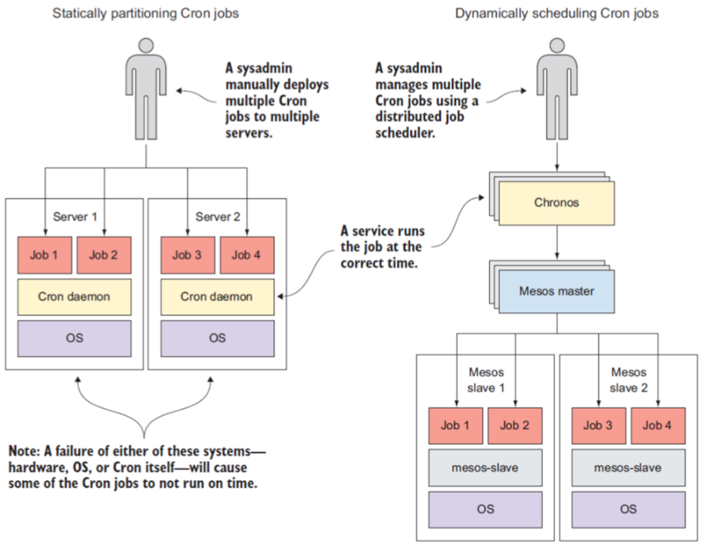


Figure 2.5 Mesos and Chronos provide a dynamic, fault-tolerant environment to run time-based jobs.

[From: Ignazio, R. Mesos in Action (Manning Publications Co., Shelter Island, NY, 2016).]

2.3.5 Marathon

Marathon is a popular open source Mesos framework developed by Mesosphere. Marathon is used for deploying long-running services and applications, both in Linux cgroups and Docker containers. It can also be considered a private platform as a service (PaaS) on which to deploy applications. Marathon can specify the resources needed for each instance of an application and number of running instances. If a Mesos slave fails, or an instance of application crashes or exits, Marathon will automatically start a new instance to replace the failed one.

Marathon also allows users to specify dependencies on other services and applications during deployment, so an application instance can't start before its database instance is up and passing health checks. Marathon contains a list of features that should satisfy the needs of most application management scenarios such as managing applications and groups of applications with dependencies and health checks, rolling application upgrades with specific capacity requirements, a powerful web interface and REST API, and high availability (using ZooKeeper for leader election and coordination).[8]

2.3.6 Spark

Apache Spark, unified analytics engine for large-scale data processing, runs on Hadoop, Apache Mesos, Kubernetes, standalone, or in the cloud. It can access diverse data sources. Spark perform task faster and more efficiently than Hadoop's MapReduce, both in memory and on disk in many cases. Spark also provide API for several programming language, including Python, Scala and Java and support streaming workloads, interactive queries and machine learning libraries, in addition to MapReduce-like batch processing. Spark can run locally. But that is useful only for development purpose, the number of CPU cores limits the number of executors. When setting up a production Spark cluster, there are two option.

When setting up statically partitioned cluster on an Infrastructure as a Service (IaaS) provider. It will be wasting money due to cloud instances sitting idle. Find-grained resource sharing can help increase system's utilization. For example, if there are two applications likes in **Figure** 2.6, Spark and Jenkins that need to run on multiple servers. Each of these system atop a general-purpose cluster manager like Mesos that allows for this sort of fine-grained resource sharing. It can share compute resources and run multiple workloads on a single Mesos slave. This will lead to better resource utilization across many machines within a modern datacenter.[8]

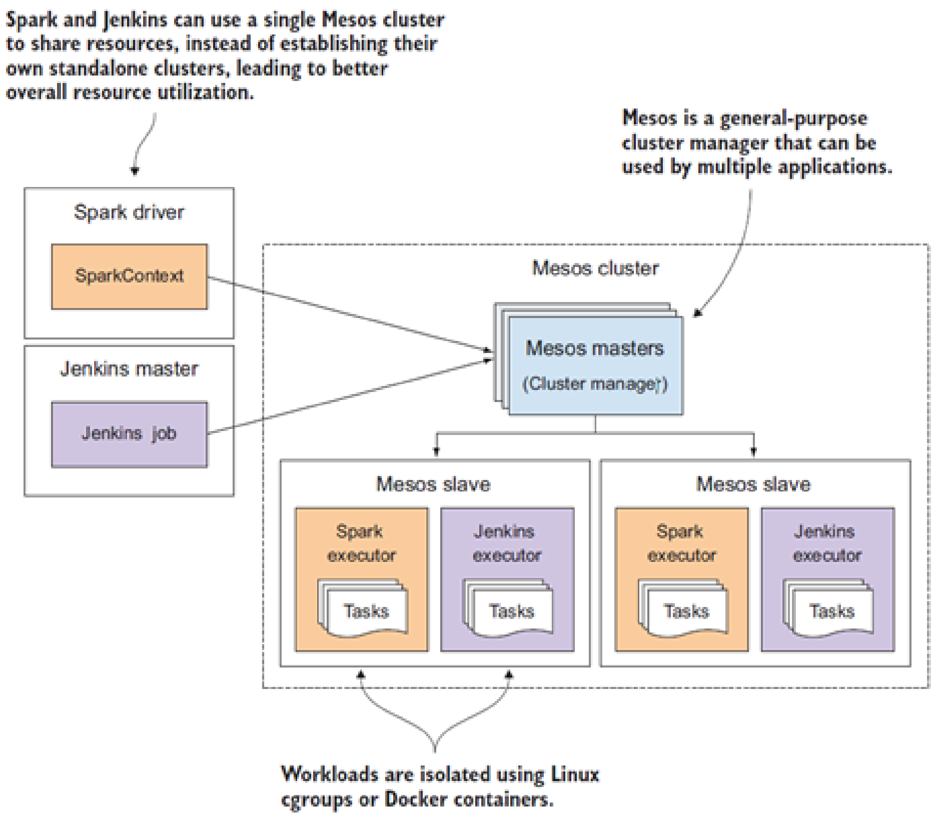


Figure 2.6 Mesos managing cluster resources for two applications.

[From: Ignazio, R. Mesos in Action (Manning Publications Co., Shelter Island, NY, 2016).]

2.3.7 Apache Kafka

Apache Kafka is an open-source stream-processing software platform. Apache Kafka is providing a unified, high-throughput, low-latency platform for handling real-time data feeds. Kafka allows user to subscribe itself and publish data to any number of systems or real-time applications. In **Figure** 2.7, producers are processes that send message to Kafka. Then, Kafka stores these messages in key-value. The data can be partitioned into different topic. And Consumers are process that can read messages from partitions. Kafka runs on a cluster of one or more server, And the topics are distributed across the cluster nodes. And partitions are replicated to multiple servers.[18]

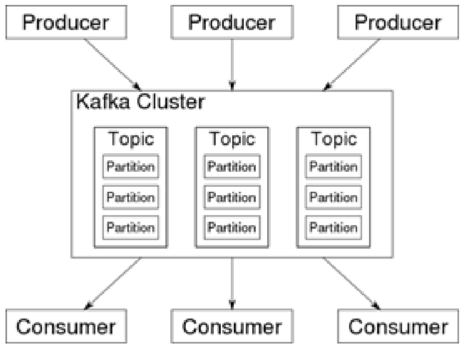


Figure 2.7 Overview of Kafka.

[From: wikipedia."Kafka".[online].Available:en.wikipedia.org/wiki/Apache_Kafka.Accessed 25 November 2020]

2.4 Related Research

The current data center management is a representative large-scale resource management and scheduling framework for clusters, liked the open-source project Mesos [8]. However, the data center environment are cluster systems and variety of submitted tasks, such as Hadoop clusters that support big data processing and Spark clusters that support in-memory computing. Mesos is a resource allocation method with no differential task type and scheduler does not consider the overall resource demand or workload, which leads to low average resource utilization and starve were a framework with a high demand on queue. Moreover, Mesos uses the DRF (Dominant Resource Fairness) algorithm for resource allocation. The DRF algorithm is the default scheduling algorithm of Mesos, the algorithm still has the disadvantage of not considering the machine performance and task type.

Many researchers have conducted relevant work. For example, in 2016, Li Y et al [10] introduced the fish swarm intelligence algorithm to dynamically adjusting the Mesos cluster resources to improve the Mesos load imbalance and resource utilization. The DRF scheduling algorithm of Mesos is extended, and in 2018, Wenbin Liu et al. proposed A X-DRF algorithm [11] based on building classifies the performance of physical machines and job type judgment classification is proposed to solve the problem of machine performance in literature, but the task type is not considered and not consider the waiting time. Compared with the original DRF algorithm, the X-DRF algorithm has higher system resource utilization rate, which is in line with the actual production rules of data centers, and provides new ideas for heterogeneous cluster multi- resource management for data center managers. In 2019, Pankaj Saha et al. developed Tromino [14], a policy driven queue manager. Tromino allows task from individual frameworks to be scheduled based on each framework's overall resources requirement and current resources consumption. Tromino reduce the impact of unfairness due to framework specific configuration and unfair waiting time due to higher resource demand in a pending task queue.

CHAPTER 3 METHODOLOGY AND DESIGN

In the previous chapter, we have explained a background knowledge that is important for better understanding of this project. Now it is the time to know an overall of our project i.e., how to design a meta-scheduler, functionality lists, and hypothesis that this project needs to answer.

3.1 Project Functionality

This project is based on the hypothesis If the dominant share and demand awareness is known by meta-scheduler, the failure job in each framework will be reduced.

This project aims to design additional architecture (meta-scheduler) of Apache Mesos to prove the hypothesis above. The user submits the task directly to the meta-scheduler, When the meta-scheduler accepts tasks, it aggregates all tasks based on data from cluster master for a better fairness and utilization.

3.1.1 Sequence diagram

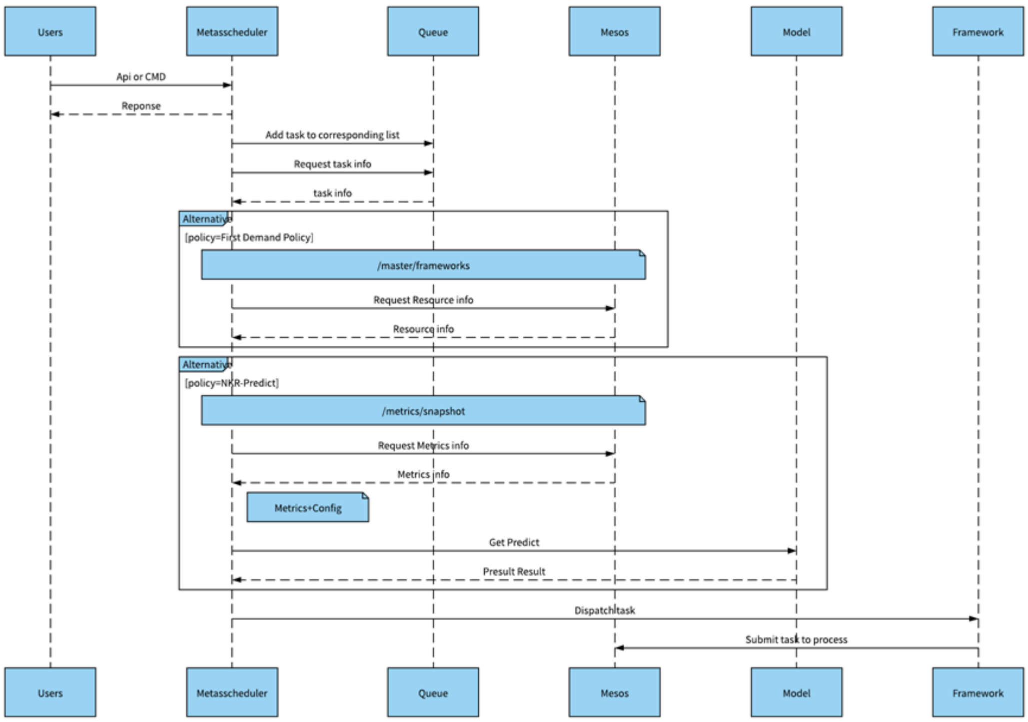


Figure 3.1 Sequence diagram of Apache Mesos

Figure 3.1 shows sequence diagram that user submits a job via command line or Application Programming Interface (API), and also specifies their framework, resources requirement, configuration script, and command to the meta-scheduler. The meta-scheduler, called NKR-scheduler, keeps track of incoming task from user and distributes each task to the queue. NKR-scheduler periodically fetches cluster and task information from Mesos Master to make a decision for task scheduling based on pre-defined policy (to be explained in Topic 3.2.2). After NKR-scheduler finishes a decision-making process, it will dispatch task directly to corresponding framework.

3.1.2 Hypothesis Testing

Hypothesis Testing was set up in order to proof that NKR-scheduler can reduce task failure in each framework. This project separates into 2 experiments because it considers both Dominant share and Demand awareness which describes more detail in **Table** 3.1. This project uses three frameworks which are Marathon, Chronos, and Spark for the test. The results were compared using 4 criteria for executing framework;

1. Normal Cluster Setting (not using NKR-scheduler)
2. NKR-scheduler – apply with Policy 1 (First Demand Share Policy)
3. NKR-scheduler – apply with Policy 2 (Success rate prediction)
4. NKR-scheduler – apply with both Policies

Table 3.1 Variable description.

Variable	Description
Dominant share	The resource that each framework uses in cluster at that point of time
Demand awareness	resource requirement of each framework

Experiment 1: Framework with different arrival rate and all tasks are identical in a resource consumption as shown in **Table** 3.2. This experiment will run 3 times consist of 1) First Demand Share Policy, 2) Success rate prediction, and 3) Both policies.

Table 3.2 Variable description.

	Number os tasks	Arrival rate (sec)
Marathon	100	5
Chronos	100	2
Spark	100	1

Experiment 2: Framework with different arrival rate and all tasks are vary in a resource consumption as shown in **Table** 3.3. This experiment will run 3 times consist of 1) First Demand Share Policy, 2) Success rate prediction, and 3) Both policies.

Table 3.3 Variable description.

	Number os tasks	Arrival rate (sec)
Marathon	100	5
Chronos	100	2
Spark	100	1

3.2 System Architecture

3.2.1 Architecture diagram

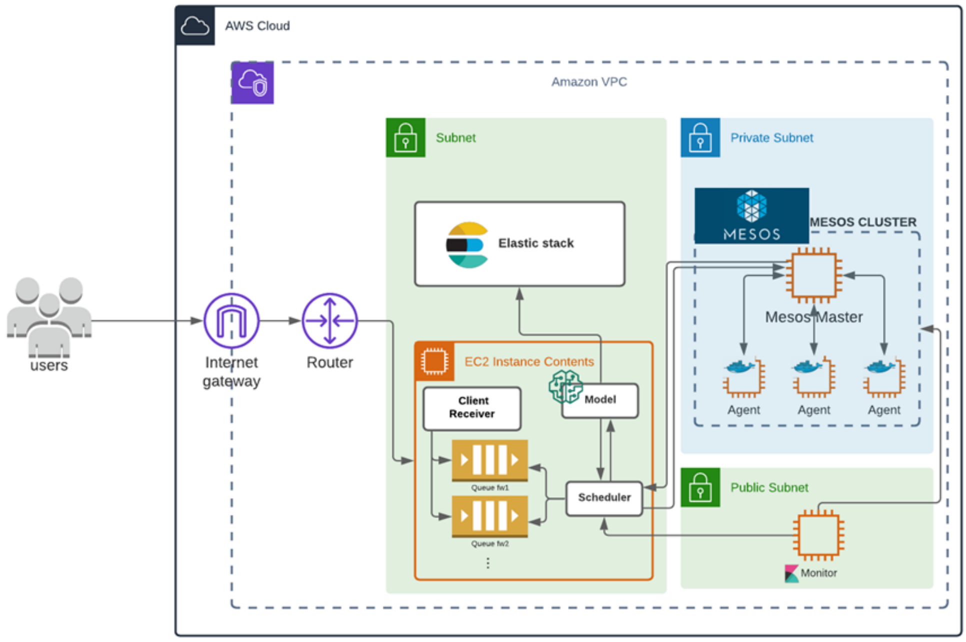


Figure 3.2 Architecture diagram.

According to **Figure** 3.2 shows architecture diagram, there are many components that this project implements. NKR-scheduler consists of three major components 1) Client Receiver, 2) Queue, and 3) Scheduler and the other components describe below in **Table** 3.4.

Table 3.4 Description of Architecture diagram.

Components	**Description**
Client Receiver	Interface for user to submit task to cluster.
Queue	The list that stores task, and information about resources requirement in each framework that register into the Apache Mesos.
Scheduler	The adaptive policy that configured by user and keeps track information about cluster from Apache Mesos.
Elasticsearch	Elasticsearch stores previous data from Apache Mesos and periodically train a new model.
Monitor	Monitor keeps track abnormality and another metrics.
Model	It stores model for predicting in case user want to use AI to schedule tasks.

3.2.2 Policy

This project designs two policies for the Meta-scheduler: 1) First Demand Policy (FDP), and 2) Success rate prediction. These policies can be extended further based on scheduling needs of users.

1. **First Demand Share Policy** We explain how FDP policy work by considering, a cluster with a total of 8 CPU and 64 GB of memory, where two frameworks (A and B) are shared completely shared resources. Each framework can have different number of tasks in its list. In example, framework a has 4 tasks each task consumes (2 CPU, 0.5GB memory) as a resource demand, and Framework B has 2 tasks with

(0.5 CPU, 1 GB memory) and task still running on cluster. Framework A has 1 task (2CPU, 0.5 GB memory), and Framework B has 2 task (1CPU, 2 GB memory) the dominants share of framework A = max (2/8,0.5/64) = 25% and Framework B = 12.5%. In this case if we apply normal scheduling policy it will dispatch task from framework B as shows in **Figure** 3.3.

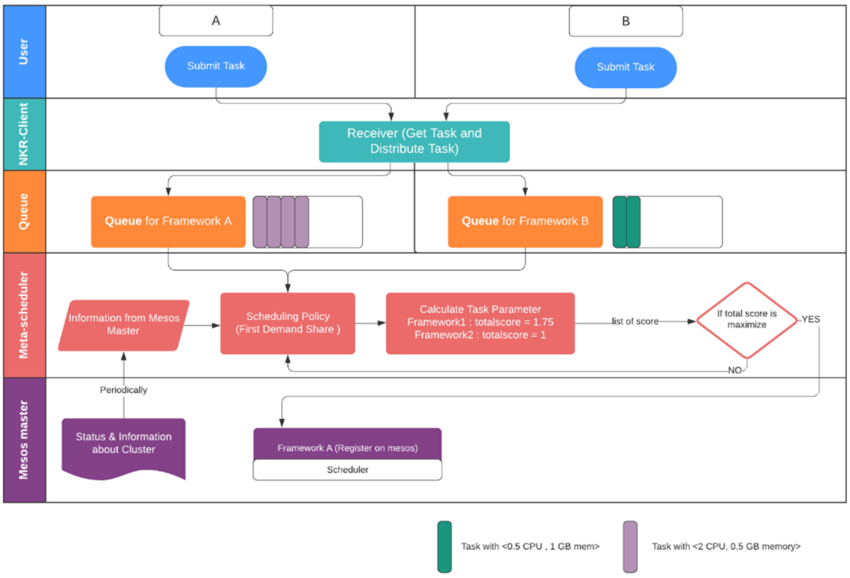


Figure 3.3 Architecture diagram.

In FDP policy, we consider both the demands of each framework and their dominant share that come from cluster monitor. Scheduling only based on the demand may cause unfairness. A framework could end up consuming the entire cluster due to its higher demand while another framework that has significantly fewer number of tasks to execute could starve for resources. Therefore, we combine both as a factor and use decision matrix in each cycle to decide which task to be dispatched. We present how to calculate each parameter in **Table** 3.5. according to **Table** 3.6, We can see highly total factor in framework A, therefore program assign a higher priority and let its corresponding dispatcher release a task

Table 3.5 Parameter Formula and Description.

Parameter name	**Formula**	**Description**
Demand Dominant (DD)	$DD_i = \max_{j=1}^{m}(\frac{n_i rd_{i,j}}{r_j})$	m available types of resources
		n_i number of tasks on list i
		$rd_{i,j}$ resource demand of type j being demand by framework rd_i
		r_j total resource of type j
Demand Share (DS)	$DS_i = \max_{j=1}^{m}(\frac{n_i}{r_j})$	m available types of resources
		n_i number of tasks on list i
		r_j total resource of type j

Table 3.6 Decision Matrix.

Framework	**DD**	**(1-DS)**	**TOTAL**
A	1	0.75	1.75
B	0.125	0.875	1

2. **Success rate prediction**There are many metrics in this policy that affect system or task performance, such as CPU utilization, free memory, storage in use, message send information, message queue length, average execution time, and etc. These metrics are able to identify anomalies (task failed, task killed, and etc.) by using the pipeline provided in **Topic** 3.3.2

   Firstly, the log data need to be pre-processed by transforming and cleaning. Secondly, the log data are clustered by K-means algorithm to separate the data that point into different metric groups or different machines. The success rate prediction of given task will use the Random Forest. Random Forest is a set of interconnected decision tree that uses the majority voting to provide classification or regression result. It is robust to noise and able to provide highly accurate prediction. [15] In each cluster, the model will be trained by active and terminated tasks information.

   In each cluster, data will be divided into 80% training dataset and remaining 20% testing dataset. The measure metric is accuracy, precision, recall and error. When users submit task, NKR-scheduler will request metric information from Mesos, allocate resource to run tasks, profile the task to data cluster and use that cluster model to predict success rates of their task shown in **Figure** 3.4.

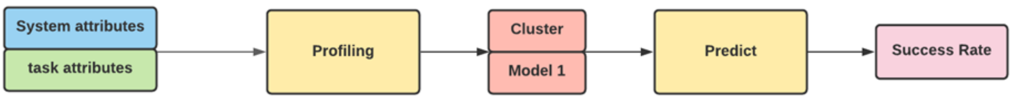


Figure 3.4 Flow Diagram of Predicting success rate.

3.2.3 Database Schema

This project implement database as a Elasticsearch and keep log every minute. It allows storing, searching, and analyzing big volumes of data quickly and schema-less. It uses a default configuration to index the data. Database schema shows in **Figure** 3.5 and describe each field in **Table** 3.7

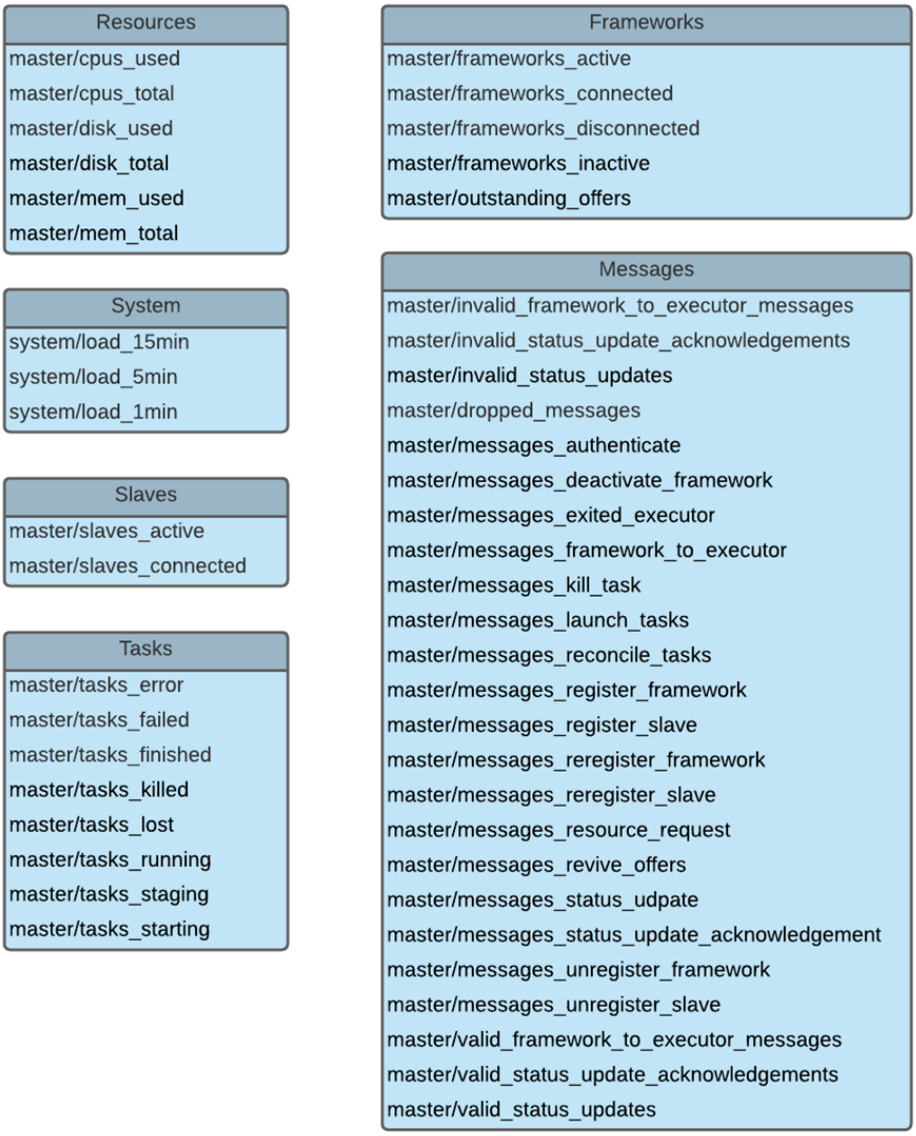


Figure 3.5 database schema.

Table 3.7 Field name and description.

<table>
<tr><th>Index Name</th><th colspan="3">Description</th></tr>
<tr><td>Resources index</td><td colspan="3">The following metrics provide information about the total resources available in the cluster and their current usage.</td></tr>
<tr><td rowspan="5">System index</td><td colspan="3">The following metrics provide information about the resources available on this master node and their current usage.</td></tr>
<tr><td>Field name</td><td>Data type</td><td>Description</td></tr>
<tr><td>Load_15min</td><td>Double</td><td>Load average for the past 15 minutes</td></tr>
<tr><td>Load_5min</td><td>Double</td><td>Load average for the past 5 minutes</td></tr>
<tr><td>Load_1min</td><td>Double</td><td>Load average for the past 1 minutes</td></tr>
<tr><td>Slave index</td><td colspan="3">The following metrics provide information about slave events, slave counts, and slave states.</td></tr>
<tr><td rowspan="10">Task index</td><td colspan="3">The following metrics provide information about active and terminated tasks. A high rate of lost tasks may indicate that there is a problem with the cluster.</td></tr>
<tr><td>Field name</td><td>Data type</td><td>Description</td></tr>
<tr><td>tasks_error</td><td>Double</td><td>Number of tasks that were invalid</td></tr>
<tr><td>tasks_failed</td><td>Double</td><td>Number of failed tasks</td></tr>
<tr><td>tasks_finished</td><td>Double</td><td>Number of finished tasks</td></tr>
<tr><td>tasks_killed</td><td>Double</td><td>Number of killed tasks</td></tr>
<tr><td>tasks_lost</td><td>Double</td><td>Number of lost tasks</td></tr>
<tr><td>tasks_running</td><td>Double</td><td>Number of running tasks</td></tr>
<tr><td>tasks_staging</td><td>Double</td><td>Number of staging tasks</td></tr>
<tr><td>tasks_starting</td><td>Double</td><td>Number of starting tasks</td></tr>
<tr><td>Framework index</td><td colspan="3">The following metrics provide information about the registered frameworks in the cluster.</td></tr>
<tr><td>Message index</td><td colspan="3">The following metrics provide information about messages between the master and the slaves and between the framework and the executors. A high rate of dropped messages may indicate that there is a problem with the network.</td></tr>
</table>

3.3 Data management

3.3.1 Which dataset is use for training?

This project uses simulation to create a large dataset, and typical usage of Mesos cluster. This project divides types of framework into 3 types 1) Application management and batch scheduling, 2) Data processing, and 3) Distributed databases and storage, but it will cover only 2 types of framework. This project implements a job simulator to submit 5 frameworks that work well with Apache Mesos and used by many organizations. Nowadays, the simulator simulates based on following example typically job, shown in **Table** 3.8 and this project will run application job according to **Table** 3.9 for one week, and the minimum task for a day is 50 for each framework.

Table 3.8 Framework of Apache Mesos.

Type of Framework	Framework1	Framework2	Framework3
1. Application management and batch scheduling	Chronos	Marathon	
2. Data processing	Spark	Dpark	Kafka

Table 3.9 Application Data.

Framework	**Application job**
Chronos	Wordcount, Kmeans, Topk, InvertedIndex
Marathon	Inline Shell Script, Docker based Application https://mesosphere.github.io/marathon/docs/application-basics.html
Spark	Wordcount, Pi Estimation, Text search http://spark.apache.org/examples.html
Dpark	Wordcount, Python program for data analysis
Kafka	Broker, Topic, Producer, Consumer

3.3.2 Transform data pipeline

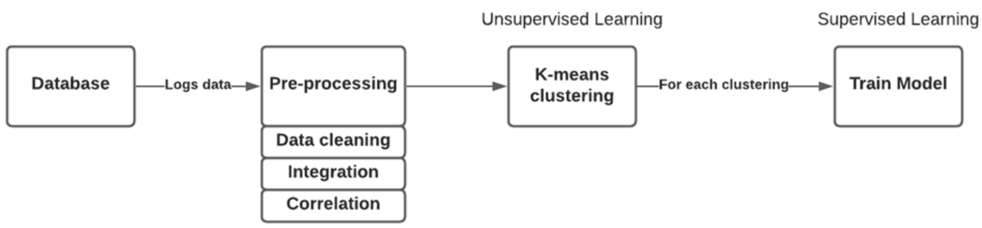


Figure 3.6 Data pipeline.

This project uses log data that mentioned before to build models by using the following steps below.

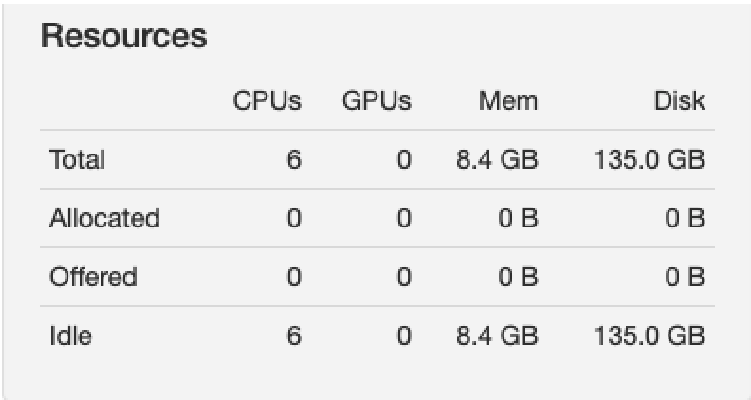
1. **Database** Collecting logs data and sending out all of data from last week.
2. **Pre-processing**
   - **Data cleaning** Delete uncompleted and unused data.
   - **Integration** Summarizing resource utilization of task, and server.
   - **Normalization** Changing value of data into the same range, without distorting differences in the ranges of values.
   - **Data selection with correlation** Finding correlation of each attribute and selecting high correlation to build a model.
3. **Data Mining** There is plenty of data and we cannot see their relationship, so this project plugs them all into K-means clustering algorithm. It will find data patterns by grouping the data into clusters. We assumed that each cluster is pointed to be a subset of the same machines, task, or framework. Each data cluster will be further investigating. Then, the data will be input those same metrics into the machine learning model.

CHAPTER 4 EXPERIMENTAL SETUP, RESULTS, AND DISCUSSION

In this chapter details about experimental setup, results, and discussion. **Topic 4.1** details about control variables in cluster for this experiment and results of simulated data. **Topic 4.2** results of data processing and model for predict success rate. **Topic 4.3** and **Topic 4.4** are the results from each policy. **Topic 4.5** is the result when used 2 policies together. Lastly, **Topic 4.6** discusses about the result of **Topic 4.3** to **Topic 4.5** and how to improve them.

4.1 Setup cluster, framework application, and parameter

For our experiment setup, we had setup a cluster inside a cloud provider which called AWS. This cluster consists of 3 nodes with 2 CPUs, 2.8 GB of memory, and 45 GB of disk for each node as shown in **Figure** 4.1. We setup this cluster with 3 widely known Mesos Framework, Spark Marathon, and Chronos, and conducted the way to test our setup by simulated data. We had randomly varied the resources of each task from the range that each task can provide.



Resources

	CPUs	GPUs	Mem	Disk
Total	6	0	8.4 GB	135.0 GB
Allocated	0	0	0 B	0 B
Offered	0	0	0 B	0 B
Idle	6	0	8.4 GB	135.0 GB

Figure 4.1 Resources of the cluster

Simulated data was created by running sample jobs in a cluster following in **Table** 4.1.

Table 4.1 Simulated task configuration.

Framework	Number of tasks	Arrival Rate (sec)
Marathon	33	2
Chronos	33	2
Spark	33	2

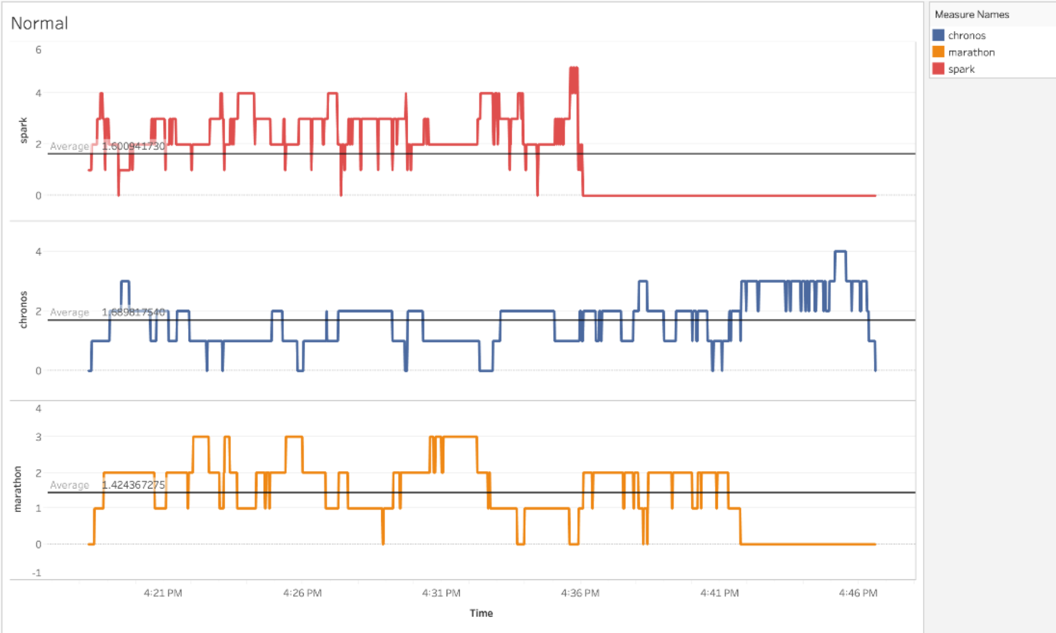


Figure 4.2 Number of tasks for each framework

We got the example of cluster metric data as in **Figure** 4.3 after run sample job in cluster and gathering cluster metric data. Each row of cluster metric data was captured every 5 seconds. , and each column of cluster metric data was an information of cluster, for example, CPU utilization, memory utilization, number of finished tasks, number of failed tasks and other information.

	A	B	C	D	E	F	G
1	_id	_index	_score	master/cpus_used	master/disk_used	master/frameworks_active	master/mem_used
2	QieZhXcBc	mesos_1	0	4.25	512	2	2,176
3	cSeZhXcBc	mesos_1	0	4.25	768	2	2,304
4	cieZhXcBc	mesos_1	0	4.25	768	2	2,304
5	oSeZhXcBc	mesos_1	0	4.25	768	2	2,304
6	oieZhXcBc	mesos_1	0	3.75	512	2	1,280
7	0SeZhXcBc	mesos_1	0	4.5	512	2	1,408
8	0ieZhXcBc	mesos_1	0	4.5	512	2	1,408
9	ASeZhXcBc	mesos_1	0	5.5	512	2	1,536
10	AieZhXcBc	mesos_1	0	5.5	512	2	1,536
11	MSeZhXcB	mesos_1	0	5.5	512	2	1,536
12	MieahXcB	mesos_1	0	4	256	2	1,152
13	YSeahXcBc	mesos_1	0	6	256	2	1,792
14	YieahXcBc	mesos_1	0	5	256	2	1,664
15	kSeahXcBc	mesos_1	0	5	512	2	2,560
16	kieahXcBc	mesos_1	0	5	512	2	2,560
17	wSeahXcB	mesos_1	0	5	512	2	2,560
18	wieahXcBc	mesos_1	0	5	512	2	2,560
19	8SeahXcBc	mesos_1	0	5.75	512	2	3,072
20	8ieahXcBc	mesos_1	0	5.75	512	2	2,560
21	ISeahXcBc	mesos_1	0	5.75	512	2	2,560
22	IieahXcBc	mesos_1	0	5.75	512	2	2,560
23	USeahXcB	mesos_1	0	5.75	512	2	2,560
24	UieahXcBc	mesos_1	0	6	512	2	3,072
25	gSebhXcBc	mesos_1	0	6	512	2	3,072
26	giebhXcBc	mesos_1	0	6	512	2	3,072

Figure 4.3 Example of cluster metric data

4.2 Model for predict success rate

4.2.1 k-means Clustering

We clustered metric data by using k-means and used elbow method to find optimal k parameter which is number of data cluster in k-means. The result is shown in **Figure** 4.4, from the graph k = 2 was chosen because at this point the distortion start to decrease in linear form. Therefore, the cluster metric data was separate into 2 clusters by k-means and the result is shown in **Figure** 4.5. But these 2 data clusters were not separated clearly, so we considered to use the whole data to build model instead.

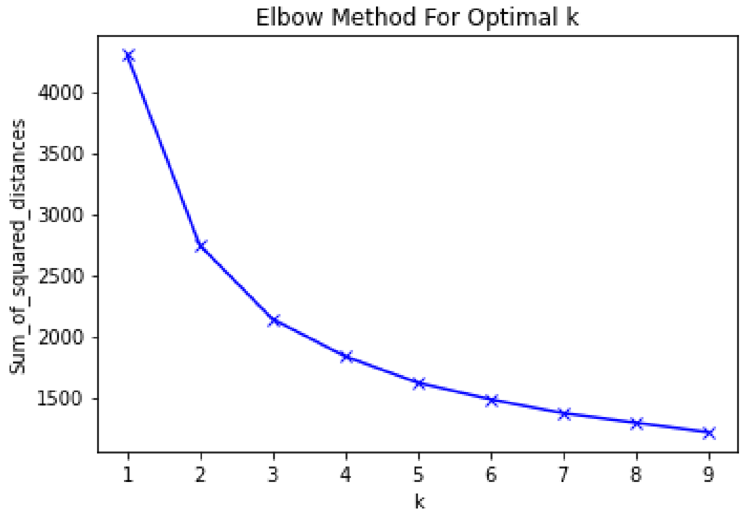


Figure 4.4 Elbow method for optimal k

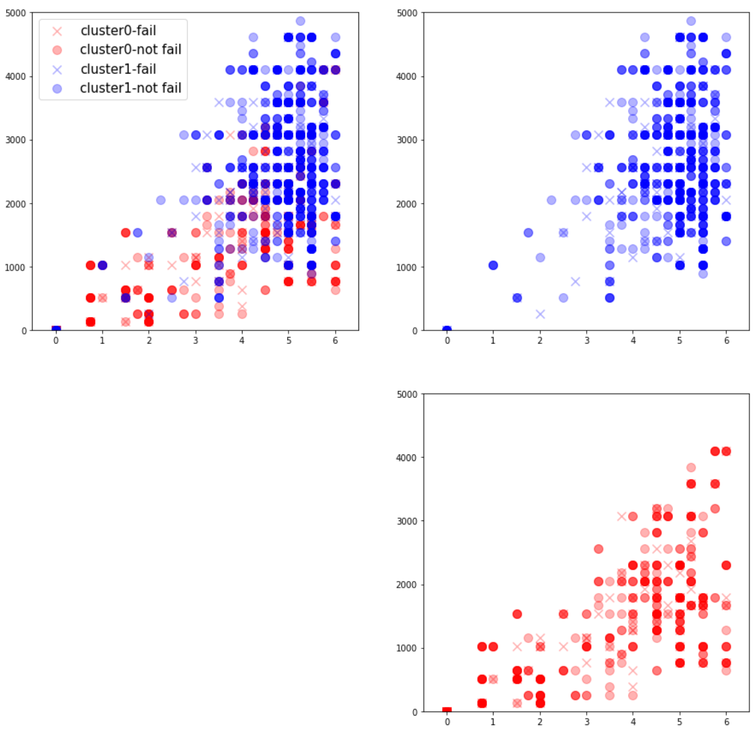


Figure 4.5 Data clustering

4.2.2 Random forest model

We trained random forest model with these following parameters; max_depth = 10, random_state = 10 and max_features = 10. Result is shown in **Table** 4.2. The accuracy of this model is 0.832, precision is 0.85 and recall is 0.98.

Table 4.2 Result from random forest model.

	Fail (predicted)	**Finish (predicted)**
Fail (actuated)	317	6
Fail (actuated)	58	0

4.3 Policy 1: First Demand Share Policy (FDP)

For the FDP, we used the same setup as mentioned in **Section 4.1**. We have developed NKR to receive tasks, and then it considers based on the first policy. We also submitted tasks as **Table** 4.3.

Table 4.3 FDP task configuration.

Framework	**Number of tasks**	**Arrival Rate (sec)**
Marathon	33	2
Chronos	33	2
Spark	33	2

The result of submitted task is shown in **Figure** 4.6.

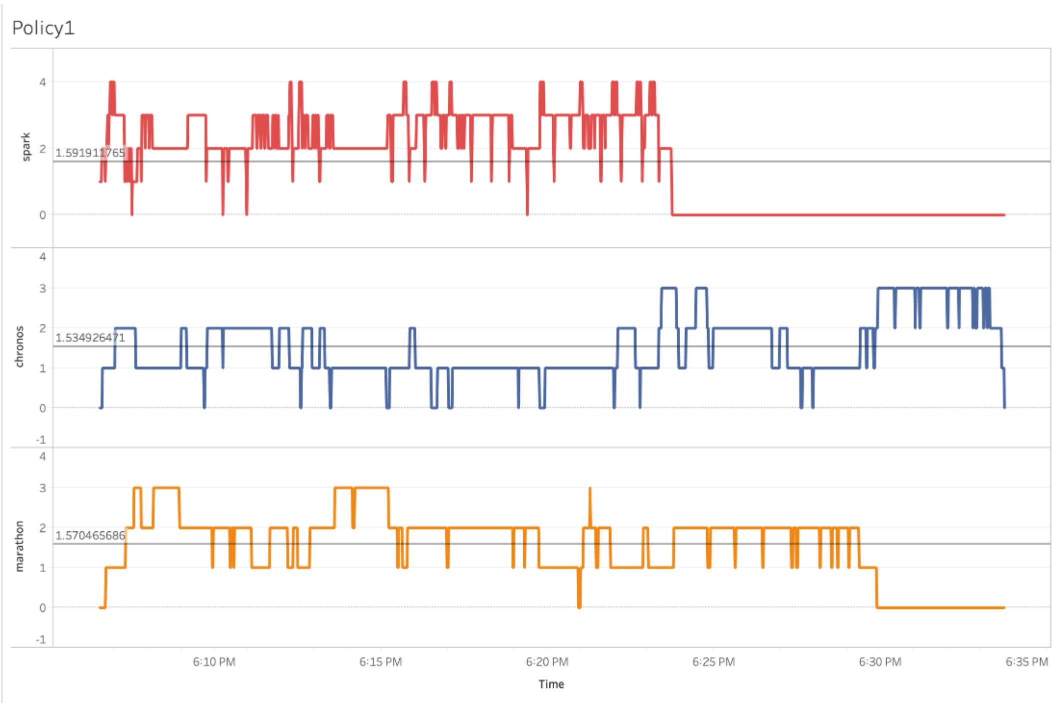


Figure 4.6 Number of tasks for each framework after using FDP

From **Figure** 4.2, Marathon and Chronos could not launch a fair number of tasks because Spark could hold on to offer more than the others. After we compared with normal Mesos, we found a little improvement of fairness as shown in **Figure** 4.6. Each framework had almost the same approximate number of tasks, which is 1.5. We also calculated different average, the value that help to compare between both policies, first policy was following with these values (1.5919, 1.5349, 1.5704). After calculated, the result is 0.1867. So, this policy can improve fairness compared to the default policy 0.0380. Then it comes to the result of fairness improvement by 79.6465 %.

We also considered other matrices after applying this policy.

1. **Failed task:** The result of failed and finished tasks for each framework before and after applied this policy is shown in **Figure** 4.7 and **Figure** 4.8. The number of Chronos failed tasks was decreased, but in the other hand, number of spark failed tasks was increased. In **Figure** 4.9 shows that growth rate of failed tasks is upward when used this policy. The slope was increased from 0.195 to 0.251. So, this policy cannot reduce the number of failed tasks in every framework and increase its failure rate.

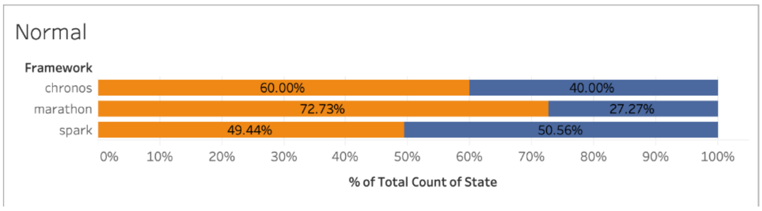


Figure 4.7 Number of failed and finish tasks before using FDP

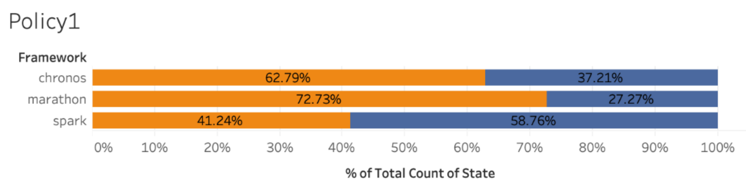


Figure 4.8 Number of failed and finish task after using FDP

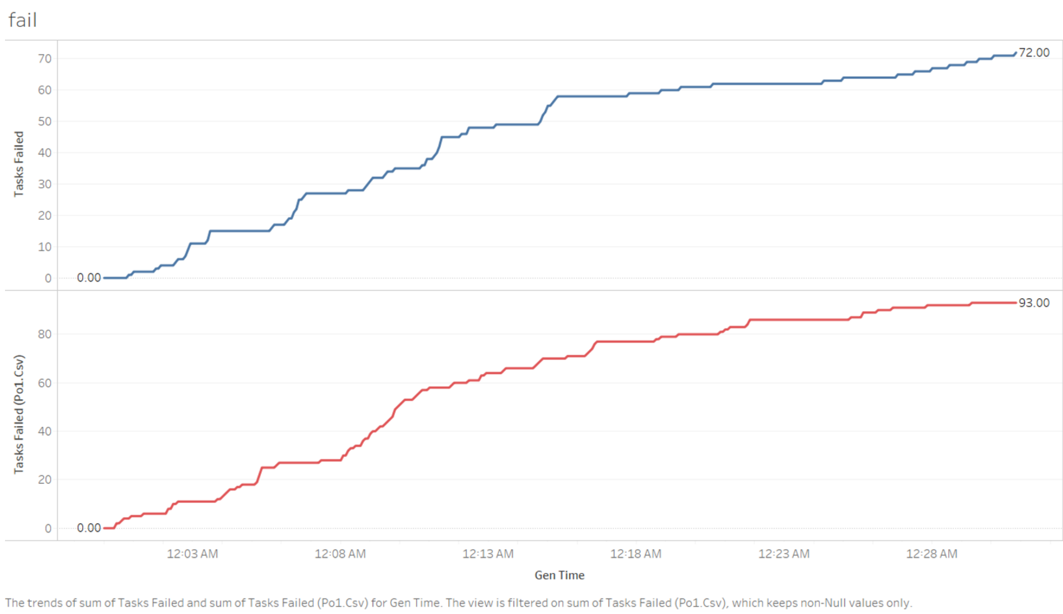


Figure 4.9 Growth rate of fail task before and after using FDP

2. **CPU and memory utilization:** The averages of CPU and memory utilization of cluster framework before and after used this policy is shown in **Table** 4.4. From the table, both CPU and memory utilization average were decreased after used this policy. The amounts of CPU and memory utilization before and after used this policy in each time is shown in **Figure** 4.10 and **Figure** 4.11.

Table 4.4 average of CPU and memory utilization before and after using FDP.

	Before	**After**
CPU	4.982	4.755
Memory	3,408	3,348

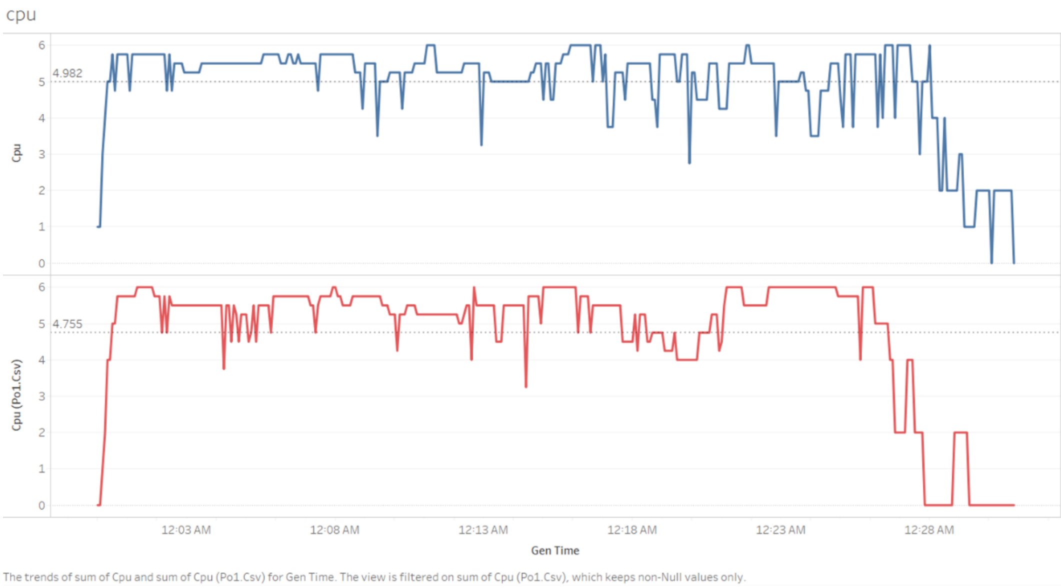


Figure 4.10 CPU utilization before and after using FDP

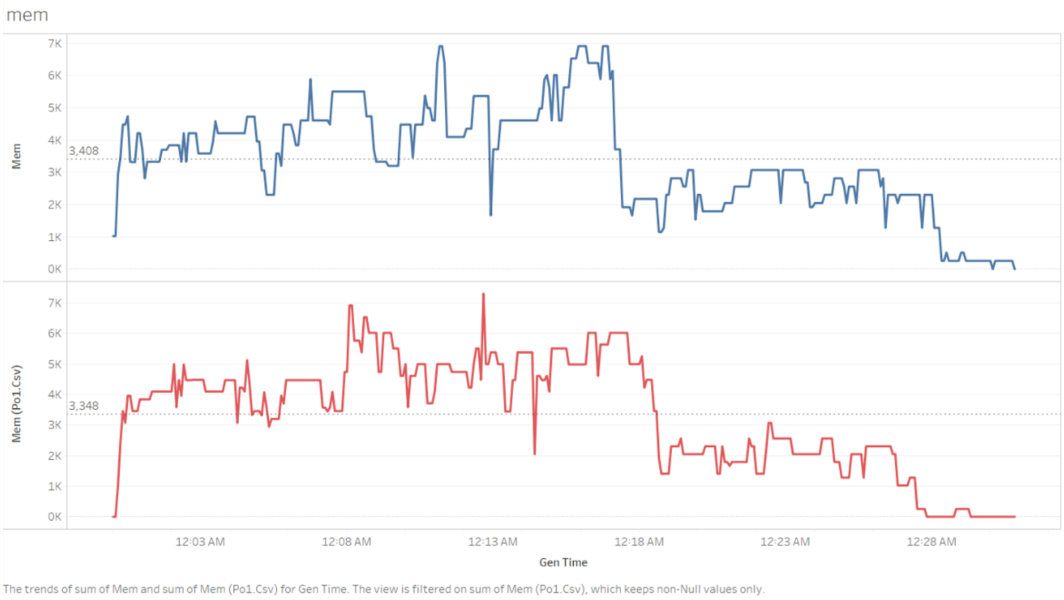


Figure 4.11 Memory utilization before and after using FDP

3. **System load:** The result of system load in this cluster before and after used this policy is shown in **Figure** 4.12. System load average before using FDP is 2.142 and after is 1.943. So, system is less busy when applied this policy.

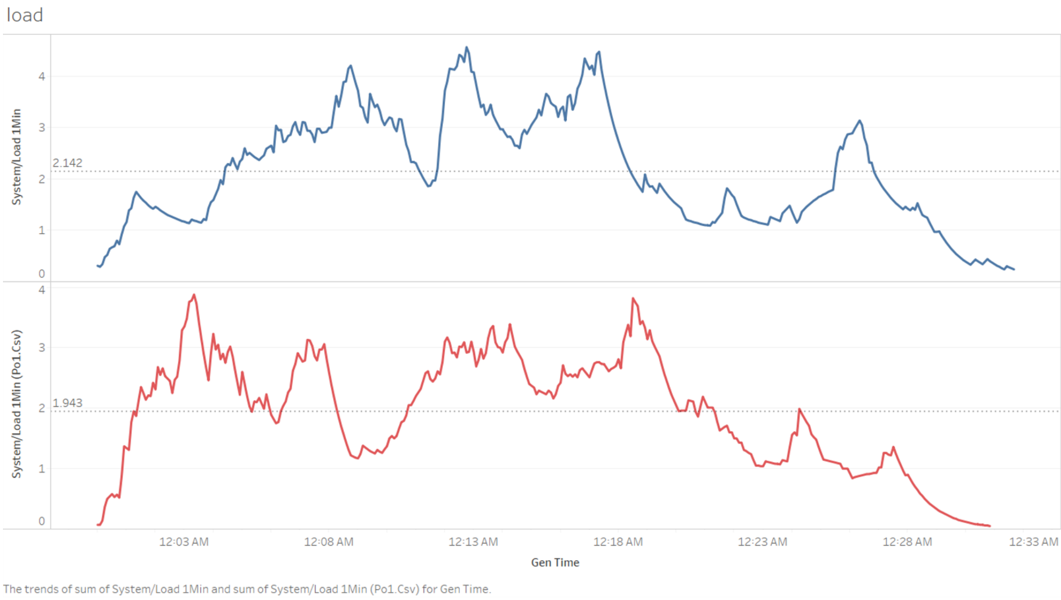


Figure 4.12 System load before and after using FDP

Therefore, Policy 1 (FDP) can improve fairness, resources utilization, and system load. By the way, the number of failed tasks is decreased only in some frameworks and failure rate is increased.

4.4 Policy 2: Success rate prediction

4.5 Policy 1 and 2

4.6 Discussion

CHAPTER 5 CONCLUSIONS

This chapter is optional for proposal and progress reports but is required for the final report.

5.1 Problems and Solutions

State your problems and how you fixed them.

5.2 Future Works

What could be done in the future to make your projects better.

REFERENCES

1. Jonas DeMuro, 2019, "What is container technology?," www.techradar.com/news/what-is-container-technology, Accessed: 2020-08-30.

2. docker, 2020, "What is a Container?," www.docker.com/resources/what-container, Accessed: 2020-08-30.

3. expertsystem, 2020, "What is Machine Learning? A definition," https://expertsystem.com/machine-learning-definition/, Accessed: 2020-08-30.

4. Jake Frankenfield, 2020, "Artificial Neural Network (ANN)," https://www.investopedia.com/terms/a/artificial-neural-networks-ann.asp, Accessed: 2020-09-05.

5. Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica, 2011, "Dominant resource fairness: fair allocation of multiple resource types," **8th USENIX conference on Networked systems design and implementation**, pp. 323–336, march 2011.

6. Ali Ghodsi, Matei Zaharia, Benjamin Hindman, Andy Konwinski, Scott Shenker, and Ion Stoica, 2011, "Mesos: A Platform for Fine-Grained Resource Sharing in the Data Center," **8th USENIX conference on Networked systems design and implementation**, April 2011.

7. IBM, 2020, "Apache ZooKeeper," https://www.ibm.com/analytics/hadoop/zookeeper, Accessed: 2020-09-05.

8. Roger Ignazio, 2016, **Mesos in Action**, Manning Publications Co.

9. Anton Kirillov, 2016, "Resource Allocation in Mesos: Dominant Resource Fairness," http://datastrophic.io/resource-allocation-in-mesos-dominant-resource-fairness-explained, Accessed: 2020-09-5.

10. Qing Liu, Tomohiro Odaka, Jousuke Kuroiwa, and Hisakazu Ogura, 2013, "A Fair Scheduling Algorithm for Adaptive Heterogeneous Resources in Data Centers," **IEICE**, pp. 872–885, April 2013.

11. Wenbin Liu, Ningjiang Chen, Hua Li, Yusi Tang, and Birui Liang, 2018, "A Fair Scheduling Algorithm for Adaptive Heterogeneous Resources in Data Centers," **the Tenth Asia-Pacific Symposium**, September 2018.

12. Hargrave Marshall, 2019, "Deep Learning," https://www.investopedia.com/terms/d/deep-learning.asp, Accessed: 2020-09-03.

13. Abueg Ralf, 2020, "Elasticsearch: What It Is, How It Works, And What It's Used For," https://www.knowi.com/blog/what-is-elastic-search/, Accessed: 2020-11-24.

14. Pankaj Saha, Angel Beltre, , and Madhusudhan Govindaraju, 2019, "Tromino: Demand and DRF Aware Multi-Tenant Queue Manager for Apache Mesos Cluster," **2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC) 63-72**, May 2019.

15. Mbarka Soualhia, Foutse Khomh, and Sofiène Tahar, 2018, "Adaptive Failure-Aware Scheduling for Hadoop," **2015 IEEE 34th International Performance Computing and Communications Conference (IPCCC)**, February 2018.

16. TheMathWorks, 2020, "What Is Deep Learning?," https://www.mathworks.com/discovery/deep-learning.html, Accessed: 2020-09-03.

17. Robbert van Renesse, Aage Kvalnes, Dmitrii Zagorodnov, and Dag Johansen, 2006, "Policies and Metrics for Fair Resource Sharing," **University of Tromsø**, January 2006.

18. Wikipedia, 2020, "Apache Kafka," https://en.wikipedia.org/wiki/Apache_Kafka, Accessed: 2020-11-25.