UNIVERSITY OF MISKOLC

FACULTY OF MECHANICAL ENGINEERING AND INFORMATICS

MapReduce Performance Analysis and Modelling of Auto-Scaling Applications Behaviours

PHD DISSERTATION

AUTHOR:
Ebenezer Komla GAVUA
MEng. in Computer Science

'József Hatvany' DOCTORAL SCHOOL
OF INFORMATION SCIENCE, ENGINEERING AND TECHNOLOGY

HEAD OF DOCTORAL SCHOOL:
Prof. Dr. Jenő SZIGETI

ACADEMIC SUPERVISOR:
Prof. Dr. habil. Gábor KECSKEMÉTI

Miskolc
2023

Declaration

The author hereby declares that this thesis has not been submitted, either in the same or in a different form, to this or to any other university for obtaining a PhD degree. The author confirms that the submitted work is his own and the appropriate credit has been given where reference has been addressed to the work of others.

Miskolc, 2023. July, 10

Ebenezer Komla Gavua

Acknowledgments

I would like to thank:

Gábor Kecskeméti my supervisor, for his rigorous mentoring and guidance into the world of cloud computing by helping me to appreciate scientific research and supporting me to reach my goals.

University of Miskolc especially, academic, administrative and technical staff of the Department of Information Technology (especially Prof. László Kovács, Dr. Samad Dadvandipour and Dr. Mileff Péter) for providing such a nurturing environment to conduct my research. To the staff of the International Students Office, I say a big thank you.

My Colleagues especially, Boucetta Sara Imene, Owais Mujtaba for sharing ideas valuable advices that helped me to publish papers and arrive at this dissertation. To the Iraqi squad! thanks a lot.

Loved ones especially, Rev. Dickson Tuffuor Sarpong, thank you for your prayers. Nii Longdon and Juliet, Anthony and Ida, Fehrs and his wife, Eugene, Caleb and Sena, Sheila, Delali, Richard, John, Kwadwo, Christian, Nana Ama, Divine, James, Franklin, Dominic, Yayra, Benedict, Reginald, Roland, Philip, Lawrence and David. Thank you all.

Management of KTU Thank your for your immense support.

Governments of Ghana and Hungary Thank you for making my dreams possible.

Family especially, Happy, Vivian, Smile, Kuklui, Innocent and Simon Sedem for their continuous support during my studies. Gad, thanks for your unflinching support. Mrs. Rebecca Afua Gavua (my mother), I am forever grateful. To my In-laws, I say thanks a lot. To my late father, Rev. E.K. Gavua, thanks for inspiring me from the other side.

My Wife and Children Marceline, Dzidzor, Dzifa and Dzinyuie. Thanks for your immense love, support and understanding. I had to leave you for many years but your love kept me going. Akpe, Köszönöm Szépen.

Dedication

Now unto the King eternal, immortal, invisible, the only wise God, be honour and glory for ever and ever. Now unto him that is able to keep you from falling, and to present you faultless before the presence of his glory with exceeding joy, to the only wise God our Saviour, be glory and majesty, dominion and power, both now and ever. Amen and Amen. 1 Timothy 1:17, Jude 1:24-25, King James Version

To My Family: God Has Done It! ICGC: Raising leaders!, Shaping Vision! Influencing Society through Christ! NSPPD: What God Cannot Do, Does Not Exist! AlphaHour: Every with Jesus is Everyday in Victory!

Table of Contents

List of Figures

List of Tables

List of Notations

AddVM This asm function attaches a job to a VM during VM selection. 115

BelongsTo The asm function that specifies the mapping between a VM and job. It is engaged during vm selection to link VMs to jobs and works together with compatible. 115

C_s Tasks state. 73, 75

Compatible The asm function that specifies the appropriate VM for a job during VM selection. It is employed together with BelongsTo. 115

$CORES$ is an integer which represents the CPU cores per node. 88

G_s captured snapshots. 75

H This notation represents an arbitary mapper with a snapshot approach implemented on it to capturing its runtime. 92

i^{th} This notation represents an arbitary snapshot capturing during the application of the snapshot capturing algorithm. 92

$InitReqFunctions$ This is an asm module replaces all the functions required for job initialisation. It is employed during job handling to show that job processing has already commenced.. 116

JobOutcome The asm function that specifies the results of an ASM run. 115

JobRequest This is an asm function is utilised to exhibit the request for jobs during the asm model Initial Phase. 115

JobState The asm function specifies the current state of a job. It could be in submitted, waiting, running, failed or done state. 112

JobTime This asm function specifies the current time of a job during task processing. A job could be in started, processing, stopped and completed state. 115

MappedJob This is an asm function is used represent the state of the jobs which must be connected to VMs. 115

N is an integer representing the number of nodes running on a cluster.. 73, 88

ProcessRequest This is an asm function is utilised to exhibit the request for VMs during the asm model Initial Phase. 115

ProcessState This asm function specifies the current state of a process during task processing. A process could be in new, ready, waiting, running or stopped state. 115

Q^H The cost incurred in implementing an approach overhead an infrastracture. In this context, the over head cost was determined on the implementation of the snapshots capturing on MapReduce. 92

Q_{avg} This is an asm function represents an average quantity of VMs in a VM pool. It is employed during job queuing for some auto-scaling mechanisms. 117

Q_i Tasks Instances. 77

Q_{max} This is an asm function represents a maximum quantity of VMs in a VM pool. It is employed during job handling for some auto-scaling mechanisms. 117

Q_{min} This is an asm function represents a minimum quantity of VMs in a VM pool. It is employed during job initialisation for some auto-scaling mechanisms. 117

QueReslist This is an asm refinement utilised for monitoring and providing the universes and functions of the job queuing phase. 116

R is an integer representing the number of reducers per node. 88

S_x Silhouette score. 81

Systemstate This function replaces processState, JobState, JobTime in Multimodes to reflect system state transitions. 116

T is an integer representing the mapred task tracker reduce tasks maximum value. 88

T_{avg} This is an asm function represents an average threshold level. It is employed during job queuing for some auto-scaling mechanisms. 116

T_c Completed Tasks. 77

$t_c^{H,i}$ The exact time a particular task processing was completed. 92

T_E^H The total execution time a task implemented on a mapper takes to process uploaded jobs. 92

T_{max} This is an asm function represents a maximum threshold level. It is employed during job handling for some auto-scaling mechanisms. 116

T_{MET} Maximum Tasks Execution Time. 77

T_{min} This is an asm function represents a minimum threshold level. It is employed during job initialisation for some auto-scaling mechanisms. 116

T_R Running Tasks. 77

T_s Tasks state. 75, 77

$t_s^{H,i}$ The exact time a particular task processing commenced. 92

The *ARESOURCE* represent the resource requirements for a JOB. 111

The *BASICJOBSCHEDULER (BJS)* This universe utilises clustering patterns to monitor VMs in in Simple auto-scaled environments. 111

The *GENERICTRACEPRODUCER (GTP)* This universe provisions set of JOBs for VM processing for specified durations in Simple auto-scaled environments. 111

The *JOB* The data submitted by a Jobhandler to be executed on a node. 111

The *JOBHANDLER* is the entity that processes traces and sends its jobs to a job launcher in the multimode autoscaling mechanisms. 110

The *JOBLAUNCHER* is the entity that emits jobs for processing for the multimode mechanisms. 111

The *PROCESS* They work together to enable applications to perform specific tasks. 111

The *SERVICE* The services performed on specific applications at a specific time on a particular node. 111

The *SIMSCALER* This universe is responsible for generating basic scaling activities in Simple auto-scaled environments. 111

The *TIME* The duration a submitted job must spend being processed by the Abstract resources. 111

The *VI* This universe is responsible for managing VMs for applications in multimode environments. 111

The VM The collection of VMs required for running the jobs in an auto scaled infrastructure. 111

TLevel This is an asm function which displays the VM utilisation threshold during job processing. 116

VM_F This is an asm function which represents any VM on the first position in the queue for job processing. 117

VM_L This is an asm function which represents any VM on the last position in the queue for job processing. 117

VmPool This is an asm function which displays the pool created autoscaling mechanism to host VMs during job processing. 117

VMPost This is an asm function which specifies the position of VMs during job processing. VMs could be in firt or last position. 117

VmRequest This is an asm module which is utilised for VM selection in Multimodes.. 116

$VmUL$ This is an asm function represents VMs utilisation levels during an asm run. 116

$VMut_{avg}$ This is an asm function represents an average VM utilisation level. It is employed during job queuing. 116

$VMut_{max}$ This is an asm function represents a maximum VM utilisation level. It is employed during job handling. 116

$VMut_{min}$ This is an asm function represents a minimum VM utilisation level. It is employed during job initialisation. 116

Y is an integer that represents the number of mappers per node. 88

Chapter 1
Introduction

Nowadays, the field of cloud computing is experiencing an increased rate in research focused on resource provisioning frameworks [107,119]. These works have encouraged the utilisations of robust and available programmable infrastructures. A typical parallel programming model that has been employed for data intensive and distributed workloads on clouds is MapReduce (MR) [176]. Researchers and organisations utilise MR for applications that process large data sets in parallel on clusters. The data are processed in a scalable, reliable and fault-tolerant manner [136].

MapReduce has several implementations designed for specific purposes. For instance, MARISSA [43] is an implementation for streaming science applications. Twister [51] allows for data access via local disks, and offers efficient support for iterative MapReduce computations. Mars [72] is a MapReduce framework on graphics processors. Mars hides the programming complexity of the GPU behind the simple and familiar MapReduce interface.

One of such implementation developed for distributed storage and processing large datasets using commodity hardware is the apache hadoop [80]. The core of Hadoop includes a distributed file system (i.e., Hadoop Distributed File System (HDFS)) and a MapReduce processor [167]. Hadoop executes the MapReduce programs written in different languages. HDFS is utilised for storing data and the provision of resources for job processing to promote I/O performance. In spite of all the benefits derived from Hadoop, the infrastructure has few challenges. One of such challenges is how Hadoop deals with tasks which require abnormally long run time. MapReduce reprocesses unusually long tasks (straggler tasks) on available nodes to finish the computation faster (as backup tasks) [9]. This phenomenon is known as speculative execution. Moreover, to ensure that challenges on cloud infrastructures (including MR) are tackled with less costs and efforts [4], researchers and developers have resorted to the application of simulations (i.e., computer and cloud simulations).

Several benefits have been derived from the application of cloud simulations [138]. Since these simulations enable the provisioning of requisite computational capacity to solve problems in a reasonable amount of time [171]. Also, the flexible nature of cloud simulations provide researcher with more options which are risk-free as compared to the real life experiments [63].

Over time, research has progressed to evaluating computer simulations and their tools to highlight their benefits and disadvantages. Mansouri et al [107] reviewed over thirty-three cloud simulators based on multi-level feature analysis of simulators in the cloud computing environment. Byrne et al [26] reviewed thirty-three simulators based on autonomous simulation platforms (including plugins and extensions that different aspects of cloud, edge and fog computing). Also, Fakhfakh [58] analyzed and compared twenty-two popular cloud simulators on their general features while Zhao [181] reviewed eleven cloud simulators on their power consumption and response time. Amongst the simulators reviewed includes CloudSim [27], CloudAnalyst [168], DISSECT-CF [84] and WorkflowSim [35]. One of the main benefits of utilising simulations is the freedom to control the environment and to provision resources to meet demands. This is reflective of what pertains in auto-scaling mechanisms (auto-scalers).

Auto-scaling mechanisms are designed to exhibit several resource provisioning behaviours including scaling vertically (add more RAM or CPU to existing Virtual Machines (VMs)) and horizontally (add more VMs) scaling [54]. This is to ensure that jobs are processed in a scalable environment. Several research efforts [104] have been carried out to analyse the resource provisioning behaviours exhibited by auto-scalers, especially when they are developed on different frameworks. Most of these research efforts applied statistical and experimental [66] approaches. However, research has shown that there are available flexible and verifiable ways to evaluate the resource provisioning behaviours of auto-scaling mechanisms [111, 131, 140].

Now, the issues presented above require a strategic direction and procedures to systematically tackle them. Therefore, an outline of research aims is required to highlight the steps to be employed to address them.

1.1 Aims of the Research

I. To devise a framework that simplifies the selection of MapReduce simulators for researchers based on vital simulator features.

   (a) The framework should enable researchers to analyse the strengths and weaknesses of simulators.

(b) The framework should comprise of MapReduce specific criteria which allows users to determine the appropriate simulators for research and development.

II. To offer a speculative execution approach for MR Hadoop that improves job performance and scales with a data centre.

(a) To devise a solution that captures task run times during job processing towards the determination of fast tasks and straggler tasks.

(b) To allow users to assess tasks behaviours on several MapReduce Hadoop setups. To allow users to select tasks which require restart or those which require rescheduling to other computers.

III. To enable users to analyse the virtual machine provision behaviours of auto-scaling mechanisms independent of previous approaches.

(a) The framework should allow users to compare auto-scaling mechanisms from multiple sources to identify existing similarities.

(b) The framework should allow users to evaluate auto-scaling mechanisms based on job execution phases.

1.2 Dissertation Guide

This guide is designed to enable readers to peruse this dissertation in a systematic manner. An outline of chapter dependencies of this dissertation is presented in figure 1.1 for the ease of navigating the dissertation.

Beginning from chapter 1.0 (i.e., introduction), readers can navigate their way to other chapters (and sections of chapters) of the document following the coloured dependency arrows. The arrows point towards the sections that must be read before navigating towards later portions of the document. The single circles represent sections in the dissertation. The double circles represent the last sections (end goals) of a particular chapter.

There are five colours (i.e., yellow, red, blue, brown, green) on the figure which helps readers to navigate towards our reading goals. Colour yellow directs readers to the first reading goal (i.e., **Contributions**) in section 7.2. Colour blue directs readers towards the second reading goal (i.e., **Resource Provisioning**) in sections 2.9 and 2.10. Colours brown and red directs readers to the third and fourth reading goals (i.e., **MapReduce Classification Framework** and **SE Approach**) in sections 3.3 and 4.3. Colour green

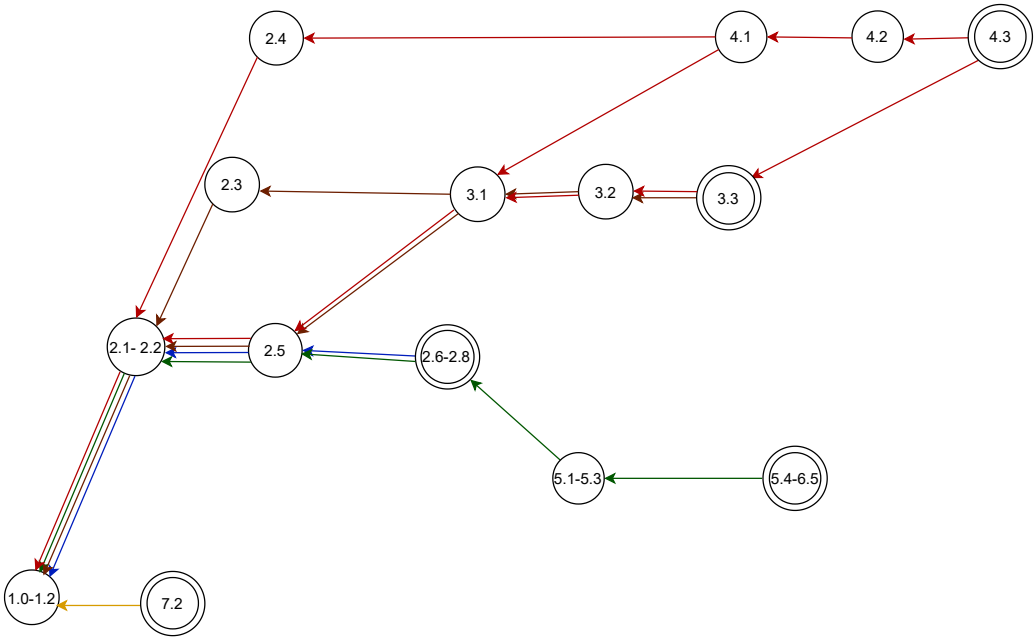


Figure 1.1: Dissertation Guide for Readers

directs readers to the fifth reading goal (i.e., **Auto-Scaling Model**) in section 5.3. The colours are not needed to under the arrows on the figure but they foster the navigation process (i.e., an additional tool).

Chapter 1.0 to section 1.2, introduces the dissertation's direction and research aims. These serve as the launch pad to navigate towards the reading goals (as shown in figure 1.1). Now, let us discuss five reading goals described below.

Contributions Beginning from chapter 1.0, a reader can navigate to section 7.2 to read the theses statements that highlights the research efforts of this dissertation.

Resource Provisioning Discussions A reader traversing from chapter 1.0 to section 2 accesses the literature review segment of the dissertation. Sections 2.1 to 2.2 presents the background information of this dissertation. It fosters the comprehension of the the design processes, and the evaluation of approaches proposed in our theses. Reading from sections 2.5 to 2.8 provides a reader with detailed information about the virtual resource provisioning content of this dissertation.

MapReduce Classification Framework A reader traversing sections 2.3 to 3.3 should see details of the processes utilized to create our first contribution to science. However, for further comprehension, a reader must

first read the literature reviewed in section 2.5 to appreciate resource provisioning on MapReduce.

SE Approach A reader traversing sections 2.4 to 4.3 should see the description of the design of our second contribution to science. Nevertheless, the reader must read the introduction of MapReduce in section 3.1 before reading section 4.1. In section 4.1, we introduce MapReduce's Hadoop speculation execution and all the necessary information required before the design of our contribution. Also, the reasons for the choice of the simulator utilized for the experiment in section 4.3 are discussed in section 3.3. Hence, for comprehension, it is recommended to read the evaluation of simulators in section 3.3 before 4.3.

Auto-Scaling Model A reader traversing sections 5.1 to 6.5 should see the design of our third contribution to science. However, the reader must read sections 2.5 before 5.1 to 5.2. The literature reviewed in section 2.5 to foster the comprehension of the research carried out in sections 5.1 to 5.2. Also, sections 2.6 to 2.8 should be read before section 5.3. Since sections 2.6 to 2.8 provides the necessary background about the formal technique (ASM theory) employed to design our model in sections 5.4 to 6.5.

Chapter 2
Background and Related Works

2.1 Introduction

This chapter discuses the background information and research works related to our dissertation. The purpose of the background section is to provide the necessary concepts and definitions to achieve the aims of our research. These includes concepts related to cloud computing models, computer simulation, MR and the Abstract State Machines (ASM). Also, issues relating to few cloud computing simulators are clarified on their application to real life and industry. These discussions helps to clarify technical terminologies that are utilised in our dissertation. Also, we reviewed research efforts related to the aims of our dissertation. Let us get move into the discussion.

Currently, a couple research efforts are gear towards the identification of appropriate cloud simulators for research. This is due to the fact high costs in utilising real clouds for testing researchable ideas. One area which has receive minimal attention is the evaluation of MR simulators. The MR infrastructure provides several benefits for parallel and distributed computing. However, limited work has been done towards the recommendation of suitable tools. Therefore, a review of literature towards the design of framework that simplifies the selection of simulators based on vital simulator features will allow users to assess the strengths and the weaknesses of MR simulators of their choice.

Moreover, the scalable and reliable nature of MR has made it the de facto parallel programming model applicable on clouds for several researchers and organisations. This feature is observable in its Hadoop open-source implementation, due to its capability to process structure, semi-structured and unstructured data. However, one feature that reminds a challenge on the hadoop framework is the its speculative execution. Therefore, the analysis of previous research efforts will foster the identification of the challenges with

the existing solutions towards the offering of an alternative approach.

Furthermore, several research activities have shown that the supply of resources to cloud frameworks during job processing, enhances efficient service delivery. This resource provision is mostly achieved via the application of auto-scaling mechanisms (auto-scalers) on cloud infrastructures (including MapReduce). However, the challenge then arises when these auto-scalers originate from different frameworks with resource behaviours that can not be identified. Hence, a literature review that focusses on the analyses previous auto-scaling efforts will foster the design of a framework that evaluates the resource provision behaviours of auto-scalers.

In addition, the evaluation of auto-scaling approaches cannot be concluded without a review of the application of formal methods on cloud computing and MR. This review focuses on assessing the flexibility and robustness of formal methods; especially the application of ASMs on clouds and other distributed systems. The review of ASMs technique will foster the design of a framework that allows users to compare auto-scaler developed on different frameworks to identify existing similarities.

In conclusion, the results from these discussions are pivotal to our dissertation, as the rest of work related to them. Let us now begin by reviewing some cloud computing models that are vital to our research.

2.2 Cloud Computing Models

Cloud Computing is a generic term for any information technology that delivers hosted services over the Internet. It is a technology that provides hardware and software resources to users based on pay-as-you-use through the Internet. Cloud computing technology has many features such as high scalability, ease of access, lower business risks and maintenance expenses, and reduced operating cost [5].

The major cloud computing service models are software as a service (SaaS), platform as a service (PaaS), infrastructure as a service (IaaS), and Container as a service (CaaS). However, we will discuss only on two models which are directly related to our research. This is because this research focuses on the analysis of the job performance improvement of the MR programming model when it is applied as a PaaS; and the analyses of the virtual machine provision behaviours of several auto-scaling mechanisms on an IaaS platform.

Platform as a service (PaaS) provides the appropriate environment for developers to create applications and software deployed through the internet without needing any infrastructure [169]. PaaS allows customers

to rent virtualized servers and attached services to execute their applications. The customer cannot control the cloud's infrastructure, such as servers, networks, storage, or OS. The service cost is determined according to data transfer per GB, usage per hour, I/O requests per million, storage use per GB, and data storage requests per thousand [15].

Infrastructure as a service (IaaS) provides the virtual infrastructure and raw hardware required to foster the creation and managing of services on storage devices and VMs via web-based services [106]. The IaaS model is a result of the evolution of virtual private servers. The provider of IaaS supplies users with a virtual server and one or more CPUs executing several operations. The VMs are rented either for a period of time as long as required. Also, the frameworks are billed depending on the duration, and additional services the user requires. The frameworks refer to the virtual servers containing several choices of virtualized computing resources (processing, storage, and network) offered by IaaS providers at standardized costs [126, 133].

These services are only possible via vital technologies that ensure efficient service delivery. These technologies allow users to access cloud resources at a low cost to expand the frontiers of research and development. In the next sub-section, we will discuss some of these technologies to highlight their connections to our research, since they provide the necessary background for our discussions.

2.2.1 Technologies behind Cloud Computing

The following state-of-the-art technologies are pivotal to our research on MapReduce and the auto-scaling of cloud resources.

Virtualization Technology Virtualization techniques are the basis of cloud computing since they render flexible and scalable hardware services. These technologies partition hardware resources for the provision of scalable computing platforms. Virtual machine technologies like hypervisor([45, 153]) enable users to create virtualised platforms for research and development. Virtual network advances, such as Virtual Private networks (VPN), support users with a customized network environment to access cloud resources. The automatic scaling (or auto-scaling) of the cloud resources to meet user demands is one of the techniques applied via virtualisation. In general, cloud resources are either scaled up or down per user demands.

Distributed Storage System A cloud storage model is a critical cloud resource provisioned by cloud computing. Some of the technologies that implement this model are network storage and distributed data systems.

The network storage systems are supported by distributed storage providers (e.g., data centres) to offer storage capacity via lease to users. The data storage could be migrated, merged, and managed transparently to end users for whatever data formats. Google File System is a good reference [47].

The distributed data system provides data sources that can be sequentially and randomly. Users could locate data sources in a large distributed environment by the logical name instead of physical locations. Virtual Data System (VDS) is a good reference [112].

2.3 Programming Model

An another essential concept required for the our research is the programming model technology. This technology enables users to access large data sets to design and create their applications. Programming model provide their own set of rules, principles, and best practices, which govern the way developers write code, manage data, handle errors, and interact with the underlying hardware and operating system [97]. MR is a programming model with associated implementations for processing and generating large data sets across the Google worldwide infrastructures [46].

MapReduce MapReduce is a programming model for data intensive computing used in Apache Hadoop. Researchers and organisations utilise MR for application development. These applications process large data sets in parallel on hardware clusters. The data is processed in a scalable, reliable and fault-tolerant manner. MR has three major parts, including Master, Map function and Reduce function. The Master is responsible for managing the backend Map and Reduce functions and offering data and procedures to them [136]. A MR application contains a workflow of jobs where each job makes two user-specified functions: Map and Reduce. The Map function is applied to each input record and produces a list of intermediate records. The Reduce function (also called Reducer) is applied to each group of intermediate records with the same key and produces a list of output records [95]. The master

node runs all the necessary services to organize the communication between Mappers and Reducers. An input file (or files) is separated into the same parts called input chunks. The chunks are passed to the Mappers where they work in parallel to provide the data contained within each chunk. As the data is provisioned by the Mappers, they have separate output; then each Reducer gathers the data partition by each Mapper, merges them, processes them, and produces the output file as shown in figure 2.1. The main phases of MapReduce architecture are Mapper, shuffle and Reducer. The Mapper processes input data which are assigned by the master to perform some computation to the produce intermediate results in the form of key/value pairs based on user-defined code [94]. In the shuffle phase, data is transferred from the Mapper disks rather than their main memories. The intermediate result are sorted by the keys so that all pairs with the same key are grouped together. The Reduce function receives an intermediate key and a set of values of the key. It merges these values together to generates an output based on the user-defined code [13]. The network is used to transfer the data from the local Map nodes to Reduce nodes [49]. Some MapReduce implementations include Spark [178], MARLA [56], BitDew [59], Themis [127], SASReduce [41], Disco [113], Phoenix [150], MARISSA [43], Skynet [78], Google MapReduce [92], Mars [72], Planet [122], LEMO-MR [57], Twister [51], DRYADLINQ [60], DRYAD [75], MARIANE [55] and Hadoop [167]. These MR implementations have different approaches to data processing.

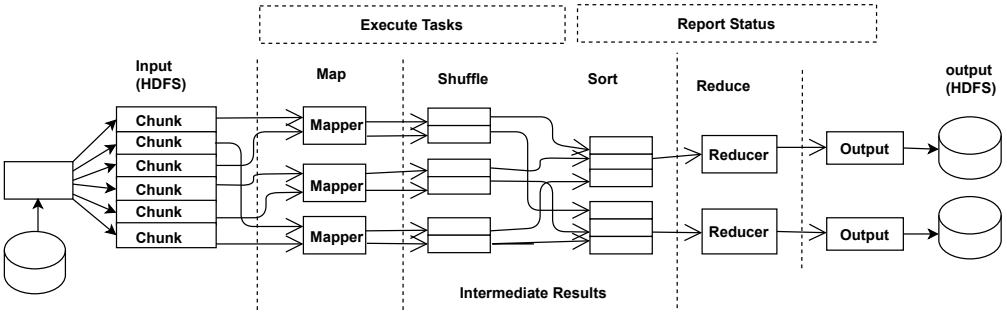


Figure 2.1: MapReduce Structure

Hadoop Apache hadoop is an open-source software framework that allows processing and managing data on multiple machines. This framework is designed to perform distributed processing with high error tolerance. The core of Hadoop includes a distributed file system (HDFS) and a MapReduce processor. In 2010, Yahoo introduced Hadoop YARN

(Yet Another Resource Negotiator). YARN added new components to Hadoop and assigned task tracker actions to the new components [167]. These include the Resource Manager, Application Manager and Node Manager.

The resource manager runs as a daemon on a dedicated machine and acts as the central authority allocating resources among competing applications of the cluster. The application master is responsible for managing all aspects of job processing, including dynamically increasing and decreasing of resources, managing the flow of execution (e.g., running reducers against the output of maps) and handling faults. The node managers are responsible for monitoring resource availability.

A critical challenge that affects data processing on MRH is speculative execution. If a task of a job requires an abnormally long execution time, the total completion time of the job is affected. Such a task is called a straggler task. MR reruns straggler tasks on a different machine to finish the computation faster. The process of diagnosing straggler tasks and assigning them to other nodes is called speculative execution [9].

The Hadoop distributed file system (HDFS) works closely with MRH for distributed storage and computation across large clusters. HDFS combines storage resources that can scale depending upon requests and queries while remaining inexpensive and within budget [147]. HDFS accepts data in any format like text, images, and videos, regardless of architecture; and automatically optimizes them for high bandwidth streaming [145].

The deployment of real clouds incurs high costs and great effort. In general, cloud simulators have been utilised to evaluate the provision of cloud resources in a flexible systematic manner. Let us now have an in-depth discussion on cloud simulations.

2.4 Cloud Simulation

Computer Simulation (CS) is defined as a hybrid technology of using computer science and technology to build simulation models and then perform experimentation on the models under several conditions. It has advantages such as high efficiency, high security, scalability, and flexibility. CS has become an important tool for design, analysis, and evaluating systems (especially complex systems), and has been playing important roles in domains

of astronautics, military, economy, medicine and entertainment, with great success [102, 175].

In general, simulation is widely used in science and industry to study a variety of problems across a wide range of domains [2]. Simulation has been applied in the area of cloud computing to allow users to solve several challenges. This has brought loads of benefits to users of cloud simulators to test their ideas before implementing them on live systems. Let us discuss the benefits of cloud simulators.

A cloud simulator helps to model several kinds of cloud applications by creating data centres, virtual machines and other utilities that can be configured appropriately, thus making it easier to analyse several models. Cloud simulators had been developed and are being actively used to conduct cloud research. These simulators varies in features like availability of GUI, licensing, base programming language, extensibility etc. The main benefits of cloud simulators are:

Cost Minimization Purchasing softwares for cloud simulators costs less when compared to buying hardware and proprietary software (operating systems, hypervisor etc). Also many simulators are available free of cost [1].

Repeatable and Controllable Experiments Experimental set ups (i.e. simulations) can be processed several times until the desire output is obtained.

Environment A simulator provides the vital environment for evaluating several scenarios under different workloads.

Do not require much of the expertise The application of a simulator does not require special skills related to deployment of cloud. The user just needs to possess the requisite programming skills for manipulating the code according to his/her needs and the rest is left to the simulator [148].

Reduced risk involved Utilising simulators (and not the real cloud) allows users to test and verify the results without the risk of financial and infrastructural loss. Also, available risks can be identified during the implementation of design or any parameter [83].

In general, the architecture of a cloud simulator is four layered. They are (*i*) Resources layer, (*ii*) Cloud Services layer, (*iii*) Application layer and (*iv*) Cloud Simulator Kernel layer as seen in figure 2.2.

Application layer allows users to define, design and submit their applications through the virtualized mediums to utilise the resources.

Cloud Services layer virtualizes the available cloud resources for provisioning user requests.

Resources layer provisions hardware elements such as CPU, memory, storage and network bandwidth.

Cloud Simulator Kernel layer consists of the libraries for managing the simulation and its parameters [107].

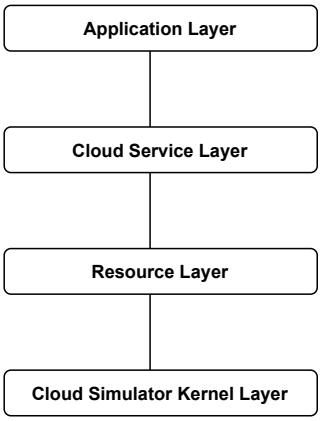


Figure 2.2: General Architecture of a Cloud Simulator

Several cloud simulators are being utilised for research and developments in real life and industry. Now, let us discuss a few of them to highlight their benefits.

CloudSim is a widely used cloud computing simulator with several extensions. It models data centers, virtual machines, service brokers, and resource provisioning methods. It provides a flexible switching between space-shared and time-shared allocation of processing elements to services. Researchers can implement cloud-based application by very less effort and time, and test the performance in heterogeneous environments [27].

By using CloudSim, researchers and industry-based developers are able to test the performance of a newly developed application or service in a controlled and easy to set-up environment. CloudSim structure is composed of three main layers: (i) User Code that provides the configuration parameters for hosts, cloudlets, VMs, number of users, and

Table 2.1: Technical Review of Cloud Simulators

Evaluation Criteria	CloudSim [27]	CloudAnalyst [168]	MDCSim [100]
Platform	SimJava	CloudSim	CSim
Language	Java	Java	Java,C++
Availability	Open Source	Open Source	Commercial
Application Model	✓	✓	✓
Communication Model	Limited	Full	Limited
Energy Model	✓	✓	✓
Platform Portability	✓	✓	✓
Documentation Available	✓	✓	×
Software or Hardware	Software	Software	Software
GUI	×	✓	×
Publication Year	2009	2010	2009
Last Updated Date	14/10/2020	15/08/2016	N/A

broker scheduling algorithms, (*ii*) Cloudsim that manages the execution of core elements such as cloudlets and data centers during simulation, and (*iii*) CloudSim core simulation engine that models queuing and communication between component [107].

CloudAnalyst is an extension of cloudSim to enable applications to manage workload descriptions including the number of users, data centers, and cloud resources along with the location of both users and data centers. CloudAnalyst can be used by application developers or researcher to determine the best strategic allocation of resources among the available cloud data centers. Data centers can be selected strategically considering the application workload and the available budget [168].

MDCSim Data Center Simulator (MDCSim) is a commercial comprehensive and scalable simulation toolbox that is used for indepth analysis of multi-tier data centers. It models the underlying hardware characteristics of data center components and estimates the power consumption

Table 2.2: Technical Review of Cloud Simulators

Evaluation Criteria	DISSECT-CF [84]	GreenCloud [88]	WorkflowSim [35]
Platform	–	NS-2	CloudSim
Language	Java	C++, OTcl	Java
Availability	Open Source	Open Source	Open Source
Application Model	✓	✓	✓
Communication Model	Full	Full	Limited
Energy Model	✓	✓	✓
Platform Portability	✓	×	✓
Documentation Available	✓	✓	✓
Software or Hardware	Software	Software	Software
GUI	×	Limited	×
Publication Year	2014	2010	2012
Last Updated Date	13/09/2022	27/03/2021	08/11/2021

of data centers. Throughput and response time are considered as performance metrics, and the topology of the data center is supplied as a directed graph by the MDCSim network package. MDCSim helps cloud users to examine different resource configurations to improve the performance of web applications while keeping the power consumption low [100].

DISSECT-CF The DIScrete event baSed Energy Consumption simulaTor for Clouds and Federations tool is used to provision an energy-aware scheduling for infrastructure clouds. DISSECT-CF supports a resource sharing framework that can model the resource bottlenecks like CPU and network. In addition, the optimization of generic resource sharing performance enhances the entire simulation. DISSECT-CF presents a more complete IaaS stack simulation. It allows to users to derive energy consumption from several resource usage counters. DISSECT-CF consists of five layers: (*i*) Infrastructure Management, (*ii*) Infrastructure Simulation, (*iii*) Energy Modeling, (*iv*) Unified resource sharing,

Table 2.3: A Summary of Strengths of Surveyed Simulators Table 1

Simulators	Strengths
CloudSim [27]	(1) It enables the modelling of data centers, virtual machines, and resource provisioning. (2) It offers a basic energy consumption model based on CPU utilization. (3) It applies space-shared and time-shared allocation policies.
CloudAnalyst [168]	(1) It presents a full GUI and geographical factors. (2) It presents a wide range of configurable factors such as data center distribution and user location.
MDCSim [100]	(1) It considers multi-tier data center structure for the study of application the performance in a scalable platform under different network loads with different tier configurations. (2) It supports power utilization and switches connected along with nodes. (3) It has low simulation overhead which is suitable for the evaluation of large-scale and three-tiered applications with varying the configuration of each tier.

and (v) Event system [84, 107].

GreenCloud [88] provides a more balanced trade-off between computing performance (CPU power) and the energy consumption of server with using three different power saving modes. Unlike common toolkits such as CloudSim [27] or CloudAnalyst [168]. GreenCloud uses aggregates, and processes information of the energy consumption in cloud data centers. GreenCloud is an extension to the network simulator NS2 and presents a fine-grained model for energy consumed through several networking components such as servers, switches, and links.

Additionally, GreenCloud takes into account the distribution of workload. The architecture of GreenCloud is a three-tier data center structure that consists of three layers: (i) access, (ii) aggregation, and (iii) core layers. The aggregation layer can facilitate increasing number of servers while keeping inexpensive switches (layer 2) in the access network and so a loop-free structure is provided.

WorkflowSim is used for modelling scientific workflows in cloud environ-

Table 2.4: A Summary of Strengths of Surveyed Simulators Table 2

Simulators	Strengths/Focus
DISSECT-CF [84]	(1) It provides a unified resource-sharing schema to enable in-data-center networking during experiments. (2) It offers energy modeling for the monitoring of the energy usage of different elements of resources such as network links and disks. (3) It determines the internal details for the definition of new physical machines, cloud topology, VM schedulers, and power states to obtain more accurate results.
GreenCloud [88]	(1) It offers enhanced capabilities for energy modeling. (2) It presents several architectures such as a Two-tier and Three-tier data center.
WorkflowSim [35]	(1) It supports a stack of workflow parser and workflow engine delay to workflow optimization techniques with better accuracy are implemented. (2) It implements several workflow-scheduling methods such as HEFT, Min-Min, and Max-Min for the comparison of algorithms in a simple way. (3) It considers task clustering and layered overhead to the workflow simulation.

ment. Workflows in heterogeneous distributed systems show different levels of overheads that are explained based on computational operations and miscellaneous works. Most simulators such as CloudSim do not consider fine granularity simulations of workflows and task clustering. Task clustering reduces the number of jobs to be executed and so execution overhead is decreased. Generally, a job may have high risk for suffering from failures since it consists of several tasks. Researchers with WorkflowSim can study the impact of job failures and the runtime performance of workflows for different clustering methods [107].

WorkflowSim is composed of several components: (*i*) Workflow Mapper that maps abstract workflows to concrete workflows, (*ii*) Workflow Engine that controls the data dependencies, (*iii*) Workflow Scheduler assigns jobs to resources, (*iv*) Clustering Engine that groups small tasks into a large job, (5) Failure Generator that injects task failures for each execution site and (6) Failure Monitor that stores the failure informa-

Table 2.5: A Summary of Limitations of Surveyed Simulators Table 1

Simulators	Limitations
CloudSim [27]	(1) It considers the limited network model (only transmission delay). Therefore, developers cannot test their application in realistic network topologies. Since it does not define switches and assumes that each VM is connected with all the others. (2) It does not consider the intra–data center communication and bandwidth sharing of links.
CloudAnalyst [168]	(1) It does not support network inside the data center and pricing model. Therefore, developers cannot evaluate the profit of providers for choosing the provider that satisfies the client's resource request with the lowest cost. (2) It has no TCP/IP implementation and offers limited support for power modeling.
MDCSim [100]	(1)) It does not support federation policy and so developers cannot evaluate their applications in the heterogeneous cloud with various domains. (2) It does not present complete network model. Therefore, developers cannot explore the characteristics of data centers in detail and model realistic network requests in terms of topology.

tion such as resource id and task id [35].

A technical review of the survey simulators is shown in tables 2.1 and 2.2. Also, the summary of strengths and weaknesses of survey simulators are shown in tables 2.3 , 2.4, 2.5 and 2.6 .

In spite of the benefits derived from the usage of cloud simulators, there are are few drawbacks.

Simplifications Simulations may lead to the oversimplification of complex systems, which can lead to inaccurate or incomplete results [8].

Data requirements Accurate simulations often require a large amount of data, which can be time-consuming and expensive to collect and process [110]. Also, simulations may not fully capture all of the interac-

Table 2.6: A Summary of Limitations of Surveyed Simulators Table 2

Simulators	Limitations
DISSECT-CF [84]	(1) It does not offer specific modules of security aspects for cloud platforms. (2) It provides a typical network model and so developers cannot define real network devices (e.g., routers, switches) and simulate various network architectures.
GreenCloud [88]	(1) It has a scalability problem since it requires very large simulation time and high memory spaces. (2) It does not provide a full traffic aggregation model and so developers cannot study the congestions control strategies for cloud data centers
WorkflowSim [35]	(1) It includes limited types of failures. Therefore, developers cannot simulate the situation when a task is not successfully sent due to network problems or workflow scheduler issues. (2) It does not consider the performance characteristic of file I/O. Therefore, it cannot obtain suitable simulation for data intensive applications since these applications involve reading or writing huge data files. (3) It supports only simple workflow techniques. Therefore, it cannot be used for other important approaches like workflow partitioning in their implementations.

tions and dependencies within a system, if absolute precautions are not taken. This can lead to inaccurate or incomplete results [151].

Model validation The process of validating a simulation model to ensure that it accurately reflects the real-world system it is meant to simulate can be challenging [139].

Therefore, these drawbacks should be considered when using computer simulations and efforts should be made to minimize their impact on the accuracy and usefulness of simulation results.

The knowledge computer simulations and cloud simulations allows users to have several options when embarking upon a research project. This provides flexibility and simplicity in their choice for cloud simulators. As discussed above, every simulator is designed for a specific purpose and cloud platform.

Researchers have carried out surveys to guide simulator users and developers towards their choice of simulators. However, concerning the MR programming model, a limited work has been done. Let us now discuss a survey about MR Simulators.

Table 2.7: Reviewed Articles for Cloud Simulators

Reference	Year	Main Goal	Weaknesses
Pan et al. [121]	2008	Evaluates key concepts of network simulators including GUI for simulation execution and Model documentation.	The review did not consider simulators in the cloud computing environment.
Wang et al. [162]	2012	Presents an analysis of eight simulators on whether simulators were workload- or resource-contention-aware.	The evaluation did not consider IoT-based applications, job reservations and scheduling policies.
Zhao et al [181]	2012	Analyses eleven cloud simulators on their power consumption and response time.	The work did not present the merits, demerits of MR simulators and their applications.

2.5 A Survey of Cloud Computing Simulators

Simulation techniques have proven to be very effective in understanding computer systems in a cost-effective and flexible environment. This is to due the fact that researchers have realised that the most affordable option for testing new ideas is the application of computer simulations. Due to this realisation, researchers have likewise applied simulations to the cloud technology which has really chalked remarkable advances in the cloud computing community.

The question that remains on the minds of many non-users is the difference between computer simulation and cloud simulation. Cloud Simulation is the integration of some mature technologies such as simulation techniques,

Table 2.8: Reviewed Articles for Cloud Simulators

Reference	Year	Main Goal	Weaknesses
Ahmed et al [1]	2014	Discusses eleven cloud simulators with nine specific criteria including federation policy, and communication model	MR simulators were not evaluated. Furthermore, the analysis did not consider major mobile cloud issues such as network access management.
Ettikyala et al. [53]	2015	Compares thirteen cloud tools with five evaluation criteria such as security, performance, and application behaviours.	The review was done based on generic criteria without highlighting the strengths and weaknesses of the simulators.
Sinha et al. [137]	2015	Provides an overview of fourteen cloud simulators with six specific criteria including availability and GUI support.	The main features of the simulators were not highlighted. Moreover, they did not mention MR simulators.
Kaur et al. [83]	2015	Analyses seven cloud tools with nine specific criteria such as simulation time and and energy model.	It does not specify the application of simulators and their architectures.

virtualization technology and web service technologies [102]. Therefore, it is pivotal say that computer simulation is an integral part of cloud simulation.

A couple of cloud technologies have been implemented as cloud simulations. A typical example for complex data processing is the MR infrastructure. MapReduce simulations executes programs in two phases, map and reduce, so that each phase is defined by a function called Mapper and Reducer. This two-phase pattern can be found in many programs and has turned into a fundamental framework for several parallel computing research [63].

There have been many studies utilizing computer simulators to investigate distributed systems. This is due to the fact that they provide the analysis of system behaviours by considering specific components under different scenarios. Over time, research has progressed to evaluating computer

Table 2.9: Reviewed Articles for Cloud Simulators Cont'd

Reference	Year	Main Goal	Weaknesses
Chernyshev et al. [37]	2018	Reviews nine Internet of Things (IoT) simulators with eleven evaluation criteria.	It does not analyse the strengths and limitations of simulators.
Azzedin et al. [14]	2019	Presents three categories of existing IoT architectures for modelling trust.	Specific features such as workload management and streaming processing required current research and tools development were not discussed.
Mansouri et al. [107]	2020	Investigates over thirty-three cloud tools based on multi-level features.	A limited number of MR tools were discussed.

simulators and highlighting their benefits and disadvantages. Also, due to the availability of numerous cloud simulators it is important that critical evaluation of simulators is done, in order to select a suitable ones for specific research [107].

Currently, classification frameworks have been proposed to assist developers in the selection of suitable simulators. Now, we shall discuss these works in order to devise a frame that simplifies the selection of MR simulators for researchers based on vital simulator features.

Pan et al. [121] evaluated the current development status of network simulators. They discussed key concepts of network simulators while analysing them. The features of the simulators were highlighted, including their current status and future development prospects. Their work formed a base for comparative studies for other distributed system simulators. These authors limited their work to four network simulators; hence, their results would require revision regarding MR-oriented computation models. Since this work did not consider the primary features required for MR.

A few research activities have been undertaken concerning classifying cloud simulators. Cloud simulators provide a conducive environment for assessing various scenarios under different workloads. Evaluations via software costs less when compared to the purchasing and testing of hardware and proprietary softwares.

A few papers [1, 83, 148] classified various simulators based on nine evaluation criteria. These authors focused on evaluating various simulators to improve simulation efficiency and cost based on three most relevant criteria. First, they identified the base platform, which refers to the pre-existing frameworks upon which simulators are built. Then, they analysed the energy model offered to evaluate the consumption of data centres. Finally, they highlighted the importance of federation policies allowing coordination among cloud service providers. However, their work did not extend to analysing the strengths and weaknesses of MR specific simulators towards the improvement of the programming model.

Moreover, Ettikyala et al. [53] compared fifteen cloud simulators with five evaluation criteria. The study evaluated cloud computing systems according to their security, performance, and application behaviours. Sinha et al. [137] evaluated fifteen cloud simulators with six criteria. They analysed the performance of cloud systems with reduced complexity. The outstanding criteria used in their classification were focused on the software and hardware features that can be simulated. These authors limited their work to popular cloud computing simulators without discussing the application of the MR computing model.

Chernyshev et al. [37] evaluated nine Internet of Things (IoT) simulators with eleven evaluation criteria. Also, Azzedin et al. [14] evaluated three categories of existing IoT architectures for modelling trust. These authors discussed current IoT trust models based on trust design parameters and their resistance to attack types. However, these authors limited their work to IoT simulators without considering workload management and streaming processing features required in a MR-specific classification framework.

Wang [162] analysed eight simulators with three evaluation criteria. The main criterion was whether the simulators were workload- or resource contention aware. Workload-aware simulators could predict the performance of a MR job that runs on a cluster when other jobs are also running. However, it does not consider IoT-based application, job reservation and scheduling policies.

The issues highlighted in the discussion above shows that the research community will welcome a framework that simplifies the selection of MR simulators for researchers and practitioners. A summary of the reviews carried out are highlighted in tables 2.7, to 2.9.

One of the challenges affecting job performance in MR and its Hadoop implementation is speculative execution. Let us discuss research efforts made towards resolving this challenge.

2.6 Prior Speculative Execution Strategies

One of the attractive features of MR is its ability to automatically parallelize a job into multiple tasks, and transparently handle job execution in a distributed setting. However, the job completion time on MR is determined by the slowest task. During job processing, some tasks of the job spends abnormally much time during execution, thus affecting the job's entire completing time. Such tasks (also known as straggler tasks) are reprocessed as *backup tasks* on available nodes. This mechanism is known as speculative execution. We review a couple of speculative execution strategies in this section.

The Hadoop Naïve Method was implemented with the Hadoop architecture. However, most of the tasks processed during runtime were detected as slow tasks and processed as backup tasks. This affected job completion because there was no improvement in job completion time after processing the backup tasks. Also, this strategy is not suitable in heterogeneous environments. Therefore, an approach that distinguishes straggler tasks from the normal tasks during job processing will ensure job performance improvements.

Zaharia et al. [179] developed the Longest Approximate Time To End (LATE) algorithm. LATE is a simple, robust scheduling algorithm that uses estimated finish times to detect straggler tasks. LATE is not suitable in heterogeneous environments. Therefore, a dynamic approach that works in all types of environments will help estimate the task runtime to ensure the improvement of job performance.

Chen et al. [33] proposed a Self-Adaptive MR Scheduling Algorithm (SAMR). SAMR uses historical information to classify nodes into the slow map- and reduce-nodes. This makes SAMR dependent on previous tasks information. Therefore, an approach that applies the information of current tasks without depending on previous nodes will be welcomed in the research community.

Sun et al. [146] designed ESAMR as an improvement on SAMR by utilising the K-means clustering algorithm to classify historical information. Therefore, the reliance of ESAMR on previous task information makes it only applicable when there is historical information. Moreover, the K-means clustering algorithm utilised was not validated to determine the straggler tasks. Therefore, an approach that is not affected by changes in dataset and validates the kmeans clustering with silhouette scores will allow users to better assess job performance. Also, the shortcomings in relation to ESAMR concerning data cluster validation and online usage are addressed in our approach.

Chen et al. [32] proposed the Maximum Cost Performance approach, which considers the cost performance of cluster computing resources to estimate the slow tasks. However, in the map phase, task satisfying data localisation executes faster than those not satisfying data localisation. This provide an unfair comparison between the tasks at the same level. Therefore, an approach that considers all tasks at the same level will ensure an appropriate estimation of task run times.

Huang et al. [99] proposed a New Speculative Execution Algorithm Based on C4.5 Decision Tree for Hadoop (SECDT) to improve predicted execution times among previous research resulting in poor job performance. However, navigating the decision tree implemented by this strategy is prone to significant overheads. Therefore, an approach that determines task run times via snapshot captures will enable the improvement of job performance.

In conclusion, existing speculative execution strategies still encounter challenges in managing straggler tasks in Hadoop. Hence, developing an approach to improve job performance could further aid the research community.

Now, job improvement on MapReduce or any cloud architecture is requires the provision of resources to meet demands. In the next section, we will discuss some auto-scaling mechanisms and how they are applied to distributed and cloud computing infrastructures (as well as MR).

2.7 Auto-Scaling Mechanisms in Cloud Infrastructures

2.7.1 Overview

The auto-scaling mechanism (auto-scaler) of a cloud system is a essential element for resource utilization enhancement, and thus reducing the infrastructure and management costs. An auto-scaling policy defines the conditions under which computer resources can be added to or removed from a cloud-based system, in order to satisfy the objectives of the application owner or the expectations of the cloud infrastructure. Auto-scaling is divided into scaling-up/-down and scaling-out/-in methods. Also, the two approaches can be defined as vertical (add more RAM or CPU to existing VMs) and horizontal (add more VMs) scaling [54].

The need to provision resources to users demands has triggered the development of more auto-scaling mechanisms. Moreover, the quest for auto-scaling cloud resources has been due to deviations in expected resources versus actual resource usage [117]. This deviation was analysed by [104]

which allowed auto-scaling techniques to be classified into demand-oriented categories in their work.

Ghanbari *et al.* [66] formulated an auto-scaling approach that exploits the trade-off between performance-related objectives and cost minimization. However, the developed model does not sufficiently analyse the algorithms generated. Since the auto-scaling mechanisms responsible for determining the accuracy of the algorithms did not have sufficient information. Yang *et al.* [174] investigated the problem of cost-aware auto-scaling along with predicted workloads in service clouds. However, the approach is only applicable to service clouds. This inherent challenge limits the strategy's extension to several cloud platforms.

Gandhi *et al.* [62] discussed the implementation of a new cloud service, Dependable Compute Cloud. However, the method applied does not offer optimal estimates of the state of processes. Therefore, an approach that models the state transition of processes will allow users to evaluate state transitions during job processing.

Saxena *et al.* [134] developed an integrated proactive resource provisioning and allocation approach. However, the approach requires further work on tasks prediction and scheduling of VMs to reduce network traffic. Therefore, an approach that allows the modelling of the auto-scaling of resources and the scheduling of VMs can improve the current design.

Al-Dulaimy *et al.* [3] developed a novel Multi-Loop Control approach, called MULTISCALER, to allocate resources to VMs based on Service Level Agreements (SLA). However, their approach is limited to the provision of platform metrics such as CPU utilisation as input for scaling. Therefore, an approach that allows the modelling of hybrid scaling will allow service providers to design their platforms to meet the demands of a larger section of users.

Ullah *et al.* [156] proposed a Cartesian genetic programming based neural network for resource utilisation estimation. However, the method utilized is not integrated with predictive scaling mechanisms for the analysis of workloads. Therefore, an approach that allows the analysis of auto-scaling mechanism with emphasis on VMs provisions will allow the authors to improve their current design for development.

So far, we have discussed the application of auto-scaling in clouds in general. Now, let us investigate the application of auto-scaling techniques to MapReduce and its Hadoop implementation. This will provide an in-depth knowledge on the flexibility and scalability of the MR parallel computing model.

2.7.2 Auto-Scaling on MapReduce and Hadoop

A few research have been carried out about auto-scaling on MapReduce and its Hadoop implementation.

Ramanathan *et al.* [125] proposed a scalability strategy of scale-out methods to obtain an accurate prediction of job completion times on MRH. However, the strategy developed was not extended to the applying machine learning on heterogeneous Hadoop clusters. An extension, when developed, could ensure more flexibility. Applying a method that guarantees correctness when designing the structure of the approach, will foster the achievement of this extension.

Ismahene *et al.* [76] proposed an auto-scaling approach that automatically allows the provision or removal of computation nodes. However, the approach does not include a scaling policy for core and nodes, which can be processed through efficient data distribution methods. This can be achieved by applying a framework that allows users to assess the scaling policies of system resources.

Hosamani *et al.* [73] proposed an approach that addresses the elastic provisioning of Hadoop clusters. However, the prediction model proposed utilized does not apply transfer learnings (reuse of a previously learned model on a new problem) to achieve better results. In order to improve the results, a framework that enables users to analyse and compare auto-scalers will allow the authors to evaluate their strategy before implementation.

Nemouchi *et al.* [116] proposed an approach based on Hadoop that automatically adjusts the computation resources depending on the workload. However, the approach does not incorporate a scaling policy for core nodes which can be processed through the use of data distribution methods. Therefore, the application of framework that allows users to examine the job execution phases of auto-scalers while adjusting nodes, will help the authors to assess their approach systematically towards the enhancement of their technique .

Most of the auto-scaling research discussed above used statistical and experimental procedures. The research community has given less attention to devising a flexible and formal approach for analysing auto scalers. An approach that can allow users to examine the job processing behaviours of auto-scalers will be welcomed by the research community. A few authors have used formal methods (especially the Abstract State Machine (ASM) model) in cloud and distributed system. They applied them to prove the feasibility of emerging cloud technologies [12]. Let us review these related works.

2.8 ASM Models for Clouds and Other Distributed Systems

Abstract State Machines represent a well-founded framework for system design and analysis introduced by Gurevich as evolving algebras [69]. They are extensions of Finite State Machines where states replace unstructured control states with arbitrary complex data [23]. ASMs provide the flexibility to analyse and design systems formally, while considering all the essential features of the application.

A few auto-scaling works have been undertaken relating to ASM modelling. LakshmiPriya *et al.* [91] developed a formal framework for an autonomous Network-Infrastructure for grids. However, the authors did not include validation techniques and refinement schemes for grids. Therefore, a model that includes formal framework with validation and refinement schemes will allow users to apply the modelling and validating process to grids.

Bianchi *et al.* [18] utilized ASM modelling to study Grid systems as a composition of interoperable building blocks. However, the architectural specifications provided requires further works on capturing user requirements for performance monitoring. Therefore, a model that provides ASM refinements on user service requests and response for vertical and horizontal scaling will be essentially required for model adaptability.

Bianchi *et al.* [19] also developed an ASM-based model for grid job management. Nevertheless, the resource dispatching policy of the model requires further works. Therefore, an model that focuses on resources provisions behaviours will enable researchers and practitioners to evaluate all aspects of distributing computing and to provide the upgrades required.

Arcaini *et al.* [12] employed ASM to formally analyse a client-server adaptivity component for clouds. However, their design was limited to communications between client-server applications. Extensions were not provided to auto-scaling mechanisms. Therefore, model that focuses on auto-scaling of resources during job processing will allow users to examine their work for improvement.

Aside ASMs, several formal methods approaches have been applied to cloud and distributed system to provision resources and to solve challenging problems in real life and industry. In the next section, we shall review a few of these approaches to highlight the role of formal methods in current research.

2.9 Formal Methods for Clouds and Other Distributed Systems

Formal methods are system design techniques that use rigorously specified mathematical models to build software and hardware systems. In contrast to other design systems, formal methods use mathematical proof as a complement to system testing in order to ensure correct behaviour. As systems become more complicated, and safety becomes a more important issue, the formal approach to system design offers another level of insurance [130].

Formal methods have been applied in cloud computing and distributed systems to develop several frameworks. An aspect of cloud computing that formal method can be applied to, is the auto-scaling of cloud resources. However, limited research has been carried out. Therefore, the application of formal methods to auto-scaling will allow researchers to use mathematical models to analyse the structure of auto-scaling mechanisms. This will foster the provision resources to meet user demands without necessarily utilising simulations.

A couple of researchers have applied formal methods to clouds (including MR) and other distributed systems. Now, let us discuss these related works in the light of our research.

Souri *et al.* [141] employed formal methods to verify their Dynamic Data Replication with Consistency approach in Data Grids. Also, Souri *et al.* [140] employed model checking techniques to verify a data replication approach in distributed systems. These techniques are directly linked to the scaling of resources to meet user demands; however the approaches proposed were not extended to automatic scaling.

Moscato [111] described a modelling profile that enables model-driven engineering analysis and verification of cloud-based services. Also, Sahli *et al.* [131] employed formal approaches to verify their cloud systems' elasticity and plasticity properties. Analysing these approaches in contrast to grid models, it may appear that the cloud models were more focused on meeting user demands. However the cloud approaches did not focus entirely on auto-scaling of mechanisms. Therefore, an approach that captures the modelling of auto-scaling of cloud resources will provide users with an alternative to the assessment of auto-scaling behaviours in clouds.

Krotsiani *et al.* [90] presented an approach for analyzing and validating cloud certification processes based on formal techniques. Chen *et al.* [34] employed model checking techniques to monitor the performance of their developed virtual machine migration model. Choucha *et al.* [39] applied formal Computation Tree Logic (CTL) based properties verification to analyse a

cloud-based infrastructure. Karthick *et al.* [81] proposed formal modelling and verification of their resource allocation algorithm for secure service migration for commercial cloud systems.

All these approaches are tackled some aspects of cloud resource provision, however, the the approaches did not focus on analysing VMs provision behaviours entirely. The introduction of an auto-scaling extension will help researchers to evaluate the job execution phases of auto-scaling mechanisms.

Some authors applied formal methods in research involving MR and its Hadoop implementation to show the wide applicability of these methods in the field of cloud computing.

Camilli *et al.* [28] presented a framework for model checking very complex systems based on iterative MR algorithms that use a fixed-point characterization of temporal operators of CTL. Lin *et al.* [101] presented the ABS-YARN framework based on the formal modelling language Real-Time ABS. Chiang *et al.* [38] adopted a Petri net (PN) to create a visual model of the MR framework and to analyze its reachability property. However, an auto-scaling extension that allow users to analyse resource provisions and usage on MRH platforms was not tackler. Therefore, implementing a formal technique to assess auto-scaler behaviours will be vital in the clouds and other distributed systems research community.

Now, in order to ensure a systematic design of our formal method approach, we will review a few concepts related to Abstract State Machines (ASM). These concepts will foster a systematic design of our ASM model that focuses on the VM provision behaviours of auto-scaling mechanisms. Also, the concepts will provision the requisite information on the validation of an ASM model.

2.10 Review of Abstract State Machine Concepts

This section discusses foundational concepts utilized in the development of an ASM model. The discussions begins with the formal definition of Abstract State Machines and definitions applied to validate and verify a model.

2.10.1 Abstract State Machine Theory

This sub-section provides a review of the theoretical background for ASMs. The review offers foundations for the sections discussing our model design. The ASM theory encompasses a formal system engineering technique that

guides software development. The ASM theory begins from software requirements capture to their implementation.

2.10.1.1 Abstract State Machines

ASMs are transition systems based on the state concept representing the system's instantaneous configuration under development, and transition rules describing the change of state [12].

Definition 1 *A transition systems M = (S, →, L) is set of states S endowed with a transition relation → (a binary relation on S), such that every s ∈ S has some s ′ → S, and a labelling function L:S → P(Atoms). Where P(Atoms) refers to the power set of Atoms, a collection of atomic descriptions [74].*

Transition systems are models with a collection of states S, a relation →, such that the system can move from state to state. While associated with each state s, the system has the set of atomic propositions L(s) which are true at that particular state.

ASM states are multi-sorted first-order structures, i.e., domains of objects with functions and predicates defined on them. ASM *transition rules* express how ASM functions are updated and interpreted from one state to the next. Therefore *transition rules* describe the system configuration changes.

The knowledge of ASM as transitions systems and their *transition rules* allows users to modelled the system state changes where required. ASMs state changes occur after ASM computations.

ASM *computation* is a sequence $S_0, S_1, \ldots, S_n, \ldots$ of states. Where S_0 is an initial state and each S_{n+1} is obtained from S_n by simultaneously firing all the transition rules which are enabled in S_n. An ASM *main rule* is a transition rule representing the computation's starting point. The *main rule* points towards the commencement of an ASM computation and ensures the monitoring of the function updates. The basic form of a transition rule is the *guarded update*:"**if** *condition* **then** *Updates*", where *Updates* is a set of functions of the form $f(t_1, ..., t_n) := t$ which are simultaneously executed when *Condition* is true; f is an arbitrary n-ary function and $t_1, ..., t_n, t$ are first-order terms. This forms the foundation of the Control State ASMs.

Definition 2 *A Control State ASM is defined as an ASM whose rules are all of the forms as in Figure 2.3: a given control state i, only one of the conditions $cond_k$ can be true, $1 \leqslant k \leqslant n$, if any; the machine executes $rule_k$, if $cond_k$ is true and changes control state from i to j_k; the state of the machine remains the same when no condition is satisfied [22] .*

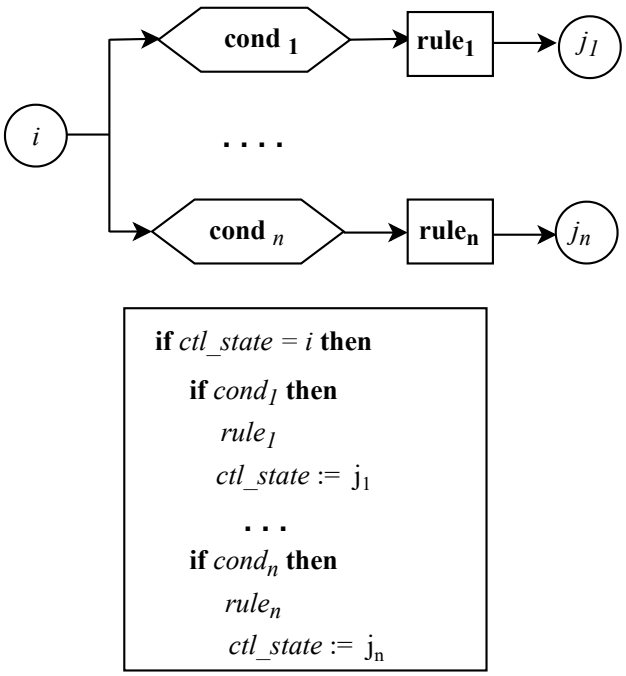


Figure 2.3: Control state ASMs

Control State ASMs are particularly useful to model system modes. The development of our ASM model begins with the creation of a ground model while utilising the model refinement technique. This allows users to appreciate the analysis and design process for model reuse.

2.10.1.2 Ground model and model refinement

A *ground model* is an ASM that can be considered a rigorous high-level system blueprint ("system contract"). It is specified using domain-specific terms that all stakeholders can understand. The ground model is *abstract*, i.e., it avoids irrelevant details necessary later for the implementation. It is *correct* and *consistent*, i.e., if it reflects the intended initial requirements and removes all the ambiguities of the initial textual requirements. However, it does not need to be *complete*, i.e., it may leave some given functional requirements unspecified. This concept is applied in creating ASM models, as it provides information on the high-level features of the design and implementation of ASM models.

This means that, all ASM models need ground model to guide their development. Once the ground model is developed, it provides the necessary platform to commence the model refinement.

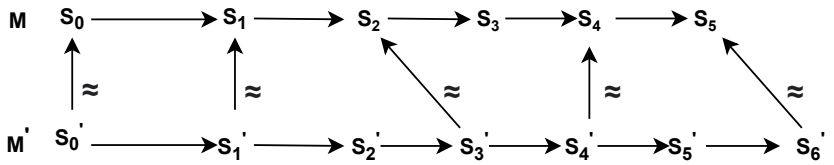


Figure 2.4: Borger's Refinement

A *model refinement* is a general scheme for stepwise instantiations of model abstractions towards concrete system elements. It provides controllable links between the more detailed descriptions of a model at the successive stages of system development. It allows progression from an abstract model to a more detailed one. ASM refinement allows one to refine either the signature (*data refinement*) or the control (*operation refinement*) [61]. This process is closely linked with the ASM refinement method defined below.

Definition 3 *The ASM refinement method is a practical stepwise method for crossing levels of abstraction to link ASM models through well-documented incremental development steps, starting from ground models and turning them piecemeal into executable code [22].*

The ASM refinement method works in conjunction with the *Börger's refinement.*

Definition 4 Börger's refinement: *Given a notion* $\equiv$ *of equivalence, an ASM* M^* *is a correct refinement of an ASM* M *if and only if for each* M^**-run* ${S_0}^*$, ${S_1}^*$,..., *there is an* M*-run* S_0, S_1, ... *and sequences* $i_0 \leqslant i_1 \leqslant \ldots$ *and* $j_0 \leqslant j_1 \leqslant \ldots$ *such that* $i_0 = j_0 = 0$ *and* $S_{ik} \equiv S_{jk}$ *for each* k *and either.*

- both runs terminate, and their final states are the last pair of equivalent states; or
- both runs and both sequences $i_0 \leqslant i_1 \leqslant \ldots$ and $j_0 \leqslant j_1 \leqslant \ldots$ are infinite.

The states S_{ik} and S^*_{jk} are the corresponding states of interest. They represent the endpoints of the corresponding computation segments for which the equivalence is defined in terms of a relationship between their corresponding points of interest, as seen in figure 2.4.

Börger's refinement is applied with Schellhorn's theorem of Forward Simulation Condition to validate and verify the ASM refinement process. Let us define Schellhorn's theorem of Forward Simulation Condition.

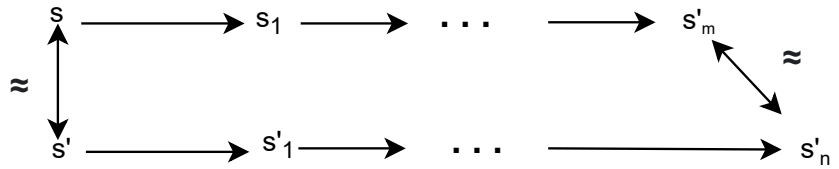


Figure 2.5: ASM state refinement steps

Definition 5 *Schellhorn's theorem of Forward Simulation Condition (FSC) [135] is defined on ASM run conditions. It states that: for two similar states $s \approx s'$, not both final, there must be m,n, both non-zero, such that M reaches a state s_m with m rule applications (i.e. $Z^m(s,s_m)$) and M′ reaches a similar state in n steps (i.e. $Z'^n(s',s'_n)$ and $s_m \approx s'\ s_n$) as shown in figure 2.5.*

Figure 2.5 shows that refinements can only be achieved in ASM via the application of rules. Therefore, to achieve equivalence of the ground model and their refinements, it is important to apply the requisite rules during the ASM computations.

The development of ground models and their refinements is hinged on the application of *universes* and *signatures*. Since they are the building blocks of an ASM.

2.10.1.3 Universe and Signatures

The formal modelling method used in this work is the ASM model (pioneered in [24]). ASM depicts states as basic sets (*Universes*) with functions and relations interpreted on them. ASMs express *Universes* with signatures to achieve model refinements.

A signature (or vocabulary) is a finite set of function names, each of fixed arity. Furthermore, ASM also contains the symbols *true*, *false*, *undef*, = and the usual Boolean operators. A state A of signature γ is a non-empty set X together with interpretations of function names in γ on X. Key Universes such as *JOB, ARESOURCE, PROCESS* and ASM functions such as *BelongsTo* have been applied in previous ground models to formalize grid computing models [115]. However, they will require redefinition to make them suitable for our work. Moreover, we developed more universes and functions to complement these ones for our work. Our model's design and analysis processes were accomplished with *universe and signatures*.

Once the model development is clearly defined, there is the need to verify the ground model and its refinements to check for correctness. In this work, the Computational Tree Logic is applied to accomplish that.

2.10.1.4 Model Verification with Computational Tree Logic

Computational Tree Logic (CTL) is a branching-tree logic, that is, its model of time is a tree-like structure in which the future is not determine; there are different paths in the future, any one of which might be the 'actual' path that is realised. This is realised via the application of CTL temporal connectives.

The CTL temporal connectives is a pair of symbols. The first is the pair of 'AE'. 'A' stands for "along all paths" (inevitably) and 'E' stands for "along at least (there exists) one path (possibly)". The second is the pair of 'X', 'F', 'G', 'U'. Where 'X' stands for "Next state", 'F' for "Some future state", 'G' for "All future state" (globally) and 'U' for "Until" [74]. The symbols X, F, G, U cannot occur without being preceded by A or an E; similarly; every A or E must have one X, F, G and U to accompany it.

Definition 6 *Let M = (S, →, L) be a model for CTL, s in S, φ a CTL formula. The relation M, s $\models$ φ is defined by structural induction on φ of a transition system.*

The following notions will be required to verify our ASM notations.

- M, s $\models$ EX φ $\iff$ for some $s \rightarrow s_1$ we have M, s $\models$ φ.
- M, s $\models$ AG φ holds $\iff$ for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \ldots$, where s_1 equals s, and all s_i along the path, we have M, s $\models$ φ.
- M, s $\models$ EG φ holds $\iff$ there is a path $s_1 \rightarrow s_2 \rightarrow s_3 \ldots$, where s_1 equals s, and for all s_i along the path, we have M, s $\models$ φ.
- M, s $\models$ AF φ holds $\iff$ for all paths $s_1 \rightarrow s_2 \ldots$, where s_1 equals s, there is some s_i such that M, s $\models$ φ.
- M, s $\models$ EF φ holds $\iff$ there is a path $s_1 \rightarrow s_2 \rightarrow s_3 \ldots$, where s_1 equals s, for some s_i along the path, we have M, s $\models$ φ.

2.11 Summary

In this chapter we discussed the background and research efforts related to our dissertation. In the background, we discussed concepts that are applicable in the later chapters of our work and also to provide clarity and focus to our dissertation.

In the related works section several issues were discussed. These can be segmented into three broad areas. Initially, we reviewed several research activities to gear towards the evaluation of cloud simulators. These researches

showed that, although a lot of efforts have been utilised to compared and analyse cloud simulators; limited work related has been accomplished in relation to MapReduce simulators. Therefore, a research towards the design of framework that evaluates the strengths and the weakness of MR simulators will be welcomed in the research community.

Moreover, we realized through research that, speculative execution remains a challenge during job processing in MapReduce Hadoop. Although, several approaches have been proposed, the existing designs had few challenges. Therefore, an alternative approach that improves job performance will enable the determination straggler tasks during job processing.

Additionally, we reviewed literature related to auto-scaling approaches. The review showed that, a couple of research efforts have been offered to ensure resource provision during data processing. However, most of the works accomplished, utilised statistical and experimental procedures. Also, the issue of evaluating the resource provisioning behaviours auto-scalers developed from different platforms remained a challenge. Therefore, a framework that allows users to analyses the VM provision behaviours of auto-scaling mechanisms independent of previous strategies towards the identification of existing similarities of auto-scalers.

Now, the issues discovered in the review literature in relation to the evaluation of cloud computing simulators can be applied in the design a framework that allows users to evaluate MR simulators and also provides recommendations for suitable ones for research and development.

Chapter 3
MaReClass Framework Design and Implementation

3.1 Introduction

This chapter focuses on the systematic analysis of cloud simulators based on selected criteria. The simulators are discussed because of the features required for their analysis and the choice of research they can be utilised for. The chapter is comprises of three broad sections. First, a brief discussion on MapReduce (MR) is provided. This lays the foundation for the research effort carried out. Second, elaborate discussions on the various strategies employed to classify MR Simulators is provisioned. Third, an in-depth analysis is conducted on selected simulators to review their strengths and weaknesses. Let us now begin the discussion on MapReduce.

3.2 Overview of the Classification Framework

In order to analyse the existing MR cloud simulation and modelling tools, a classification framework was proposed. The classification framework consists of a set of criteria used to evaluate and classify several MR simulators to highlight their strengths and weaknesses. Also, the framework provides a systematic approach to assist developers and researchers to identify appropriate MR simulators for research and development. The criteria in the framework were derived from the functional requirements of simulators. The framework was designed in four steps.

- Step 1: Identification of the features of cloud computing simulator through Systematic Literature Review (SLR).

- Step 2: The previously identified list was reduced to a more MR specific criteria via the analysis of MR themes.
- Step 3: Selection of simulators with features that identified with MR through SLR.
- Step 4: Systematic assessment of the previously identified list of simulators based on the MR specific criteria.

The first step produced thirty cloud computing themes from literature which is shown in table 3.1. These themes were the most discussed topics (i.e. keywords) related to cloud simulators. The themes were culled from over fifty cloud computing research papers. Also, the research papers were chosen via academic search engines and online libraries.

The same procedure (step 2) was utilised to refine the thirty themes to thirteen themes most relevant to MR simulations as shown in figure 3.1. The refining process focused on the most cited themes (keywords) amongst the initial list shown in table 3.1.

The thirteen themes were utilised as criteria to evaluate MR specific simulators. The evaluation showed that the absence of any of the thirteen criteria rendered detailed MR simulations incomplete. Figure 3.1 offers an overview of the selected criteria while highlighting the general and specific MR requirements. Also, the figure shows that the generic themes (requirements) support more simulators. However, the specific themes (requirement) support less number of simulators. Let us discuss the thirteen MR specific themes which were used as our evaluation criteria.

3.2.1 MapReduce Computing Model

The MR computing model enables programmers to focus on computational logic rather than the low-level programming details. This programming model allows researchers and organisation to meet their big data analysis requirements. For a simulator to be classified as supporting this feature, it should be capable of accepting data that meets the basic requirement suitable for MR computations. This means that the simulator should be capable of decomposing data sets into many independent sub-datasets for processing a particular task [98]. Also, simulators should be capable of processing huge amount of data with parallel, distributed algorithms on a simulated cluster of commodity machines [42]. Moreover, a simulator must be able to compute applications with high data intensity.

Table 3.1: List of Cloud-Oriented Themes

SN	Cloud Computing Themes		
1	Discrete Event-Based	16	Simulator's Behaviour
2	Performance Optimization	17	Network Topology Choices
3	Real MR Deployment	18	Network-Packet Level Simulations
4	Data Replication & Locality	19	Theoretical Algorithms
5	MR Computing Model	20	Single-State MR Computing
6	High-Intensity Workload	21	Network Link model
7	Completion Time Estimation	22	Hadoop Cluster Dynamic Behaviours
8	Evaluating Designs Decisions	23	Infrastructure Implement.
9	Workloads Mgt. & Resource Allocation	24	Best Job Config. Estimations
10	Stream Processing	25	Para. Tunings & Scaling Analysis
11	Optimal Parameter Config.	26	Frame Mgt. Config. & Tech.
12	Workloads Mgt Issues	27	Data Intensity MR Applications
13	Performance Analysis	28	IoT-Based Applications
14	Application Mgt. Config.	29	Experimental Environments
15	Job Scheduling Algorithms & Decisions	30	Replay of Production Clusters

3.2.2 Infrastructure Implementation

Infrastructures spread across several data centres controlled by the organization or a third party, such as a co-location facility or cloud provider [123]. A simulator should be able to represent these infrastructure constructs (i.e.

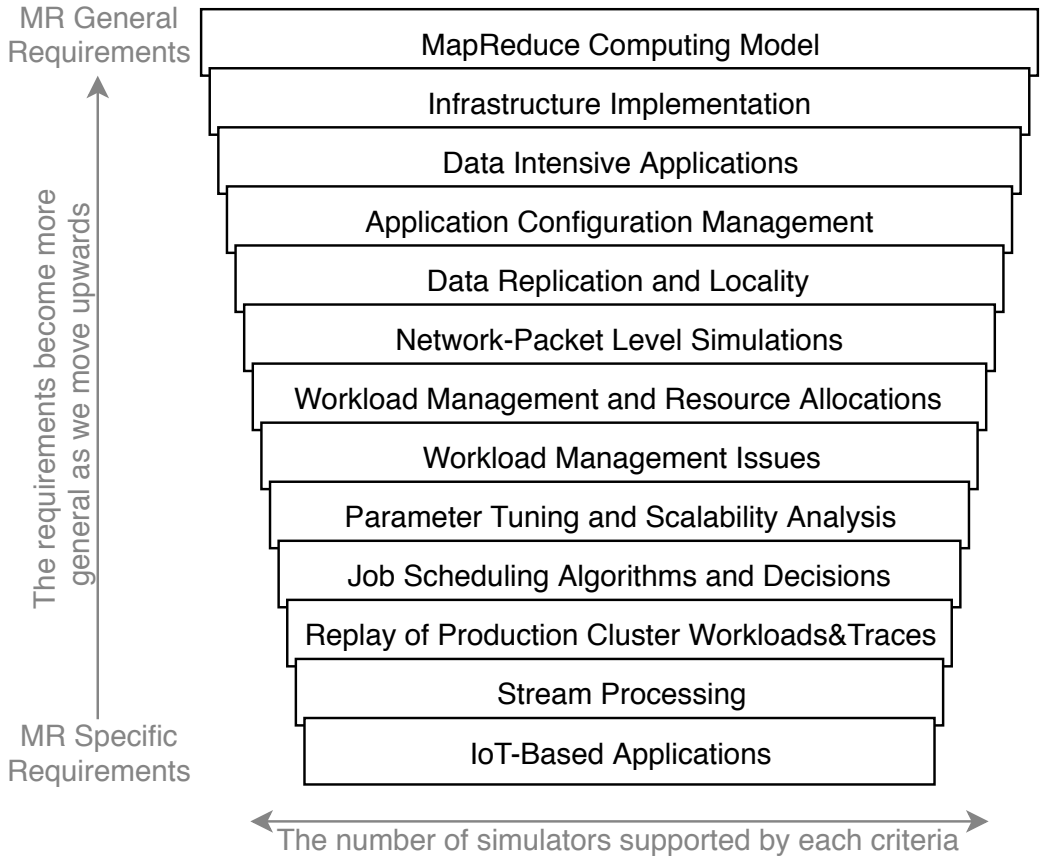


Figure 3.1: Criteria for Classification of MapReduce Simulators

basic build blocks). In general, simulators utilised Virtual Infrastructures (VI) for infrastructure implementations. A VI is a software-based IT infrastructure being hosted on another physical infrastructure and meant to be distributed as a service as in cloud computing's infrastructure as a service (IaaS) delivery model. It provides organizations, particularly smaller ones that cannot afford to build their own physical infrastructure, access to enterprise-grade technology such as servers and applications. The distribution is often done via the cloud, meaning over large networks such as the internet.

Moreover, some simulators are built upon existing simulation frameworks. The features of the existing platform are inherited. For instance, the MRSG simulator is built on SimGrid and MR-CloudSim is built on CloudSim. Simulators in this category should have an underlying infrastructure model that can interface with the MR framework.

The main features to consider for cloud simulators concerning this theme

includes the cloud types, device interfaces, compatibility, deployment requirement, and development support [25].

3.2.3 Data Intensive MapReduce Applications

This criterion refers to the capability of simulators to run applications that scale out on simulated environment to meet the demands of more than the expected inflows of data [52]. The efficient of MR feature can be achieved via the consideration and application of vital properties. These include (*i*) to accommodate large quantities of data and manipulate them efficiently; (*ii*) efficient programming model for data computation and analysis of data; (*iii*) scalable underlying hardware and software; (*iv*) the simulator should be reliable.

For data intensive applications to foster high productivity, it is important that the simulated system is scalable and the parameters of the applications are appropriately tuned. The simulated system in this context refers to the environment running the experiments.

3.2.4 Application Configuration Management

ACM allows a user to create templates to modify and manage application configurations associated with server applications. ACM enables users to update, and change configurations from a central location. This ensures that applications are accurately and consistently configured [86]. ACM automates the configuration and reconfiguration of the servers and virtual machines (VMs)in the cloud, eliminating the requirement to manually change or configure the servers, or write automation scripts.

The main requirements for ACM are (*i*) It should enable users to access any subset of versions of an object or to choose a version based on specified properties (*ii*) It should enable users to configure an object based on specified properties of its components [82].

3.2.5 Data Replication and Locality

Data replication is the process of storing data at multiples locations. In Hadoop, data locality means the computation is close to where the actual data resides on the node instead of moving extensive data to computation [143]. This feature is crucial to jobs provision during data processing.

Also, simulators should be able to perform certain functions before qualifying for this category. These include: (*i*) continuous replication with many

recovery points; (*ii*) cross-platform replication (i.e., disk to cloud and vice versa); (*iii*) replication of synchronized data with zero data loss.

Continuous replication with many recovery point: The distribution of data across large scale databases is prone to system down times. These down times affects the quantities of data stored cross various nodes. It is imperative that continuous data replication is always effected to foster updated data on all storage points.

Cross-platform replication: Cross-platform replication focuses on the utilisation of Internet Small Computer Systems Interface (iSCSI) networking standard to set up shared-storage network on cluster computers for user access. It is imperative that data snapshot replication between local devices and the cloud is implemented effectively, to prevent unsynchronized data stored across platforms.

Replication of synchronized data: The allocation of resource during task processing ensures that algorithms can effectively execute their functions. Hence, synchronized data foster high throughput for system processes.

3.2.6 Network Packet-level simulation

This is used extensively for protocol design and evaluation. Simulator designers utilise events to model real life expectations. Events in a packet level simulation represent actions associated with processing a packet such as transmitting the packet over a link. Thus the execution time in a packet level simulation is proportional to the number of packets that must be processed. This ensures that designs are assessed before implemented.

The factors to consider for this criteria are (*i*) Simulator performance (*ii*) The amount of memory required (*iii*) The amount of computation time.

Simulator performance It is convenient to use the number of Packet Transmissions that can be simulated per Second of wallclock time (or PTS) as the metric to specify simulator speed. This metric is useful because given the PTS rate of a simulator, one can estimate the amount of time that will be required to complete a simulation run. This is possible if one knows the amount of traffic that must be simulated, and the average number of hops required to transmit a packet from source to destination.

Table 3.2: Classification Framework

Evaluation Criteria	HSim [103]	SimMR [159]	SimMapRed. [152]	IoTSim [180]	WaxElephant [129]	BDSS [6]
Data Intensive App.	✓	✓	✓	✓	✓	✓
Para. Tun. & Scal. Anal.	✓	✓	✓	✓	✓	✓
Workload Mgt & Res. Alloc.	✓	✓	✓	✓	✓	✓
Job & Task Sch. Alg. & Dec.	✓	✓	✓	✓	✓	✓
Network-Packet level Simul.	✓	✓	✓	✓	✓	✓
Data Replication & Locality	✓	✓	✓	✓	✓	✓
Workload Mgt. Issues	✓	✓	✓	✓	✓	✓
Infrastructure Implement.	✓	✓	✓	✓	✓	✓
Application Config. Mgt.	✓	✓	✓	✓	✓	✓
Stream processing	×	×	×	✓	×	×
Rep. of Pro. Cls. Wk. & Tr.	✓	✓	×	×	×	✓
IoT-based Applications	×	×	×	✓	×	×
MapReduce Comput. Model	✓	✓	✓	✓	✓	✓

The amount of memory required The simulator requires a certain amount of memory to represent the state of each node, memory requirements increase at least linearly with the number of nodes.

The amount of computation time The amount of time required to complete a packet level simulation usually increases in proportion with the amount of traffic that must be simulated [105].

3.2.7 Workloads Management and Resource Allocation

workloads require effective management to forestall processing challenges. The challenges are (*i*) which processors to assign them, (*ii*) which scheduling algorithm to utilize in the assignment and (*iii*) the estimation of their processing duration. Resource allocation could often starve services if the distribution is not managed precisely [108].

Furthermore, simulators in this category should possess workload management and resource allocation strategies. Resource allocation strategy is an essential feature in the cloud computing environment as scarce resources have to be put to optimal use and utilization. In the cloud computing environment Resource Allocation is the process of allocating to the user applications according to the requirements and usage. A resource allocation strategy for simulators should be based on such criteria as, application requirements, service level agreements (SLA), over-provisioning and under-utilization of resources. However, it should avoid the following criteria:

Resource contention situation arises when two applications try to access the same resource at the same time.

Scarcity of resources arises when there are limited resources.

Resource fragmentation situation arises when the resources are isolated. There will be enough resources but adequately to allocated to the needed application.

Over-provisioning arises when the application gets surplus resources than the demand.

Under-provisioning arises occurs when the application is assigned with fewer numbers of resources than the demand [11, 160].

3.2.8 Workload Management Issues

Workload management issues are closely related to the previous criterion and arise while running MR applications. The main issues are scalability of application platforms, data locality, and replication [10].

Data distribution imbalances represent the presence of workload management issues. In order to meet this feature, it is vital to assess how all the workloads are distributed during data processing. The assessment is achieved by implementing an efficient load-balancing algorithm to monitor workloads during data processing. Also, there are challenges with monitoring which data centre has more data.

3.2.9 Parameter Tunings and Scalability Analysis

This criterion refers to adjusting specific parameters of a model or algorithm upwards or downwards. The adjustments are made to achieve an improved result in the cloud infrastructure. Also, scalability analysis determines whether a proposed new algorithm is capable of increasing its capacity with increasing demand. The algorithm capacity is the maximum workload an algorithm can handle on a given device while still meeting its SLA (Service Level Agreement) [103]. Simulators should be capable of supporting changes in algorithms and performing scalability analysis to be considered as supporting this criterion

Moreover, key parameters list should be measured and increased when needed for a simulator to be considered in this category. Additionally, the simulator itself should be able to scale with the increasing complexity of the simulated system. The following factors can determine the availability of this scalability analysis criteria. They are: load-, space-, space-time and structural scalability [20].

Load Scalability A system has load scalability if it has the ability to function gracefully, i.e., without undue delay and without unproductive resource consumption or resource contention at light, moderate, or heavy loads while making good use of available resources.

Space Scalability arises when there are limited A system or application is regarded as having space scalability if its memory requirements do not grow to intolerable levels as the number of items it supports increases.

Space-Time Scalability A system has space-time scalability if it continues to function gracefully as the number of objects it encompasses increases by orders of magnitude. A system may be space-time scalable if the

Table 3.3: Classification Framework Cont'd

Evaluation Criteria	MRSim [71]	MR-CloudSim [79]	MRSG [89]	Mumak [114]	HaSim [85]	MrPerf [164]
Data Intensive App.	✓	✓	✓	✓	✓	✓
Para. Tun. & Scal. Anal.	✓	×	✓	×	✓	×
Workload Mgt & Res. Alloc.	✓	✓	✓	✓	✓	×
Job & Task Sch. Alg. & Dec.	✓	×	✓	✓	✓	×
Network- Packet level Simul.	✓	×	✓	✓	✓	✓
Data Replication & Locality	✓	×	✓	✓	✓	✓
Workload Mgt. Issues	✓	×	✓	✓	×	×
Infrastructure Implement.	✓	✓	✓	✓	✓	✓
Application Config. Mgt.	✓	✓	×	✓	✓	✓
Stream processing	×	×	×	×	×	×
Rep. of Pro. Cls. Wk. & Tr.	×	×	✓	✓	✓	×
IoT-based Applications	×	×	×	×	×	×
MapReduce Comput. Model	✓	✓	✓	✓	✓	✓

data structures and algorithms used to implement it are conducive to smooth and speedy operation whether the system is of moderate size or large.

Structural Scalability We think of a system as being structurally scalable if its implementation or standards do not impede the growth of the number of objects it encompasses, or at least will not do so within a chosen time frame.

Once the issues relating to this criterion has been tackled, an MR Simulator should be able to manage the issues of execution traces, production workloads and resource allocation effectively.

3.2.10 Job and Task Scheduling Algorithms and Decisions

Job scheduling is the process of allocating system resources to many different tasks. Also, it determines which job to take from the queue and the amount of time to allocate for the job. This ensures that all jobs are carried out fairly and on time. Task scheduling in MRH involves assigning appropriate tasks to the correct MR task [67].

Additionally, a simulator must be able to determine the following to qualify in this category: (*i*) CPU utilization; (*ii*) throughput of all processes; and (*iii*) turnaround time of various processes. However, if the requisite *cresources* are not provisioned, the scheduling algorithms may not achieve high throughput.

3.2.11 Replaying of Production Cluster Workloads and Execution Traces

Workload traces are collected to track specific actions performed on MR systems. They provide valuable information for troubleshooting MR issues and performance [158]. A simulation's efficiency could be measured by a production workload's replays and execution of traces. Since the nature of an execution trace could determine the results obtained from an experiment.

The role clusters workloads and execution traces play are crucial to determining the appropriate MR simulator to be selected for research. Realistic workloads allows cluster operators to run simulations with realistic inputs and to amplifying the benefit of MapReduce simulators. Also, it allows MR researchers to have a better understanding of the strengths and limitations

Table 3.4: Technical Evaluation of MR Cloud Simulators

Evaluation Criteria	Mumak [114]	MRCloudSim [79]	BDSS [6]	SimMR [159]	HDEXT [64]	SimMapRed. [152]
Platform	-	Cloud-sim	Cloud-sim	-	Simgrid	SimJava
Language	Java	Java	Java	Java	C	Java
Availability	Open Source	Open Source	Open Source	N/A	Open Source	Open Source
Energy Model	×	×	×	×	✓	✓
Platform Portability	✓	✓	✓	✓	✓	✓
Documentation Available	✓	✓	✓	N/A	✓	✓
Publication Year	2009	2012	2020	2011	2016	2011
Last Updated Date	19/09/2009	N/A	26/06/2020	N/A	30/06/2022	20/09/2013

of proposed optimization techniques. Workloads are expected to be representative, that is, the synthetic workload should reproduce from the original trace. These include: the distribution of input, shuffle, and output data sizes (representative of data characteristics), the mix of job submission rates and sequences, and the mix of common job types.

The primary properties observable in simulators in this category are (*i*) data characteristics, (*ii*) job submission patterns, and (*iii*) common jobs. Data characteristics refers to the distributions of input, shuffle, and output data sizes of the synthetic workloads, which are representative and reproduces from the original traces. The Job submission-rate per time unit is reproduced only if the length of each sample is longer than the time unit involved. And a representative workload should have the same frequencies of common jobs as the original trace [36].

3.2.12 Stream Processing

This allows users to query a continuous data stream. The technology quickly detects conditions within a short period of receiving the data. Stream processing is also known as real-time/streaming analytics and complex event processing [118, 144].

In order to simulate streaming applications, the following properties must be considered. They are (*i*) Data Source and Transfer (*ii*) Latency (*iii*) Lifetime (*iv*) Time/Order (*v*) Dynamism (*vi*) Processing.

Data Source and Transfer describe the location of the data source in relation to the stream processing application. The data source can be external (e. g., an experimental instrument) or internal to the application (e. g., the coupling of a simulation and analysis application on the same resource). Output data is typically written to disk or transferred via a networking interface.

Latency is defined as the time between arrival of new data and its processing.

Throughput: describes the capacity of the streaming system, i.e. the rate at which the incoming data is processed.

Lifetime Streaming applications operate on unbounded data streams. The lifetime of a streaming application is often dependent on the data source. In most cases it is not infinite and limited to e.g., the simulation or experiment runtime.

Time/Order: Constraints defines the importance of order while processing events.

Dynamism is variance of data rates and processing complexity observed during the lifetime of a streaming application.

Processing This characteristics describes the complexity of data processing that occurs on the incoming data. It depends e. g. on the amount of data being processed (window size, historic data) and the algorithmic complexity [31].

3.2.13 IoT-based Applications

IoT-based applications refer to systems that build on and utilize typical IoT connections. IoT-based applications use machine learning algorithms to analyze massive amounts of connected sensor data in the cloud. The requirements surrounding the IoT applications are broad and very dependent on the type of sensor/application being implemented. IoT covers a wide range of applications using different standards and technologies to serve a large number of applications. These applications have different network requirements, different node distribution and different mobility scenarios. These connections include RFID, Wi-Fi, Bluetooth, and ZigBee. Also, this technology allows comprehensive area connectivity using many technologies such as GSM, GPRS, 3G, and LTE. These systems enable seamless information sharing about the condition of things and the surrounding environment with people, software systems and other machines [96]. The main requirements for IoT-based applications are (*i*) Application Requirement (*ii*) Big Data Processing Requirement (*iii*) Network and Processing Infrastructure Requirements.

Application Requirement The simulator should thus allow modelling of different IoT-based applications depending on used big data processing platforms such as MR. Since an IoT-based application generally processes large data sets stored in clouds after being collected from different devices.

Big Data Processing Requirement It is mandatorily required for a proposed simulator to meet the big data processing requirement. Depending on various IoT-based applications, the simulator should offer the capability that uses different big data processing technology to support batch processing or stream processing on the big data. Also, it should allow modelling and simulating the execution of multiple jobs simultaneously in a scalable manner as it happens in the real world.

Table 3.5: Technical Evaluation of MR Cloud Simulators

Evaluation Criteria	IoTSim [180]	WaxElephant [129]	MRSim [71]	MrPerf [164]	HSim [103]	MRSG [89]
Platform	CloudSim	-	SimJava	ns–2, DiskSim	-	Simgrid
Language	Java	Java	Java	Python	Java	C
Availability	Open Source	-	Open Source	Open Source	N/A	Open Source
Energy Model	✓	✓	✓	×	×	✓
Platform Portability	✓	✓	✓	✓	✓	✓
Documentation Available	✓	×	✓	✓	N/A	✓
Publication Year	2017	2012	2012	2009	2013	2013
Last Updated Date	08/06/2018	N/A	16/01/2015	3/8/2010	N/A	29/11/2013

Table 3.6: A Summary of Strengths

Simulators	Strengths/Focus
MrPerf [164]	(1) It helps in designing new high performance MR setups and optimizing existing ones. (2) It evaluates proposed designs and the topologies of clusters.
MR-Cloudsim [79]	(1) It provides a more accessible and cheaper way to validate MR operations.
Mumak [79]	(1) It simplifies task execution without modelling the shuffle and sort phase.

Network and Processing Infrastructure Requirements An IoT-based application requires different types of storage that are commonly ambient in cloud-based data centres to store content from the various devices. Thus, a storage layer should be modelled to simulate storage (such as Amazon S3, Azure Blob Storage, Hadoop HDFS), retrieval of any amount of data and subject to the availability of network bandwidth. It is obvious that accessing files in storages at run-time incurs additional delay for IoT-based application execution. This is due to the latencies between the nodes and storages when transferring the data files through the IoT network. Hence, the design of network between nodes and storage is required to model the aforementioned delay [87].

Now that the evaluation criteria have been discussed, let us classify the MapReduce simulators.

3.3 MR Simulators Classification frame work

The themes discussed in subsections 3.2.1 to 3.2.13 are utilised as evaluation criteria to analyse and classify MR simulators. This section comprises two subsections. First, it starts with classifying thirteen simulators. Second, recommendations for simulator developers and other researchers in the field are provided. Let us begin the discussion of the MR simulators employed for the classification framework.

3.3.1 Classifying The Simulators

The MapReduce classification (*MaReClass*) framework displayed in figure 3.1 is applied to evaluate selected MR simulators as shown in tables 3.2 to 3.3.

Table 3.7: A Summary of Strengths

Simulators	Strengths/Focus
SimMR [159]	(1) It allows the tuning up of parameters like schedulers and job queues. (2) It can assess various what-if questions and help avoid error-prone decisions.
SimMapreduce [152]	(1) It evaluates the performance of MR applications under different scenarios. (2) It provides essential entity services that can be predefined in XML format.
IoTSim [180]	(1) It supports resource allocation by modelling large scale multiple IoT applications to run simultaneously in shared cloud data centres.

The MR simulators were chosen via academic search engines and online libraries after a systematic literature review (step 3). My analysis is done to highlight the strength and weakness of each simulator and to recommend the suitable ones for research.

Let us discuss the reviewed simulators (step 4).

3.3.1.1 MrPerf

The MrPerf simulator was designed as a tool for MR infrastructure. It helps in designing new high performance MR setups and optimizing existing ones.

Architecture: The structure of MrPerf shown in figure 3.2 includes input configuration that provides a set of files, and processes different processing modules (readers). Also, these are responsible for initialising the simulator. The ns-2 driver module provides the interface for network simulation. Similarly, the disk module provides modeling for the disk I/O. All the modules are driven by the MapReduce Heuristics module that simulates Hadoop's behavior.

Analysis: As a planning tool, it evaluates proposed designs and the topologies of clusters. It can be utilised for making MR deployment far easier via reduction in the number of parameters that have to be hand-tuned [164]. MrPerf captures various aspects of MR setup, and utilises the information to predict expected application performance. The MR architecture enables it to simulate task transfers to data nodes hence satisfying the packet-level simulation requirement. As a design tool, it allows users to configure objects based on specified properties. Although based on Hadoop on, MrPerf

Table 3.8: A Summary of Strengths Cont'd

Simulators	Strengths/Focus
HSim [103]/ Hasim [85]	(1) It allows the computing of nodes with varied processing capabilities to parallelly utilize MR applications. (2) It allows packet-level simulations through the transfer of tasks to node.
WaxElephant [129]	(1) It can synthesize workloads and execute them based on statistical characteristics of workloads. (2) It can load real MR workloads derived from the historical log of Hadoop clusters, and It can replay the job execution history.
MRSim [71]	(1) It models the behaviour of data mining algorithms in hadoop environments. (2) It analyses the behaviours of Hadoop job completion times and hardware utilization.
MRSG [89]	(1) It is able to reproduce real executions of MR accurately. (2) It simulates MR with different cluster configurations.
BigDataSDNSim [6]	(1) It offers holistic modelling and integration of MR BDMS-based models that are compatible with SDN network functions. (2) It provides a framework for researchers to quantify the impacts of MR applications in terms of a joint-design of host and network.

does not simulate vital aspects of the platform, such as speculative execution (which is, the reprocessing of straggler tasks during job executions) [165].

MrPerf cannot process streaming media, interface with IoT-based applications, tune parameters for large-scale setups with many variable, and handle workload management issues. Also, it cannot handle reservation and scheduling schemes.

My *MaReClass* framework has shown that MrPerf is not capable of supporting most MR features. Therefore *MaReClass* does not recommend MrPerf for MR research.

3.3.1.2 MR-CloudSim

MR-CloudSim implements the bare-bone structure of MR on the CloudSim environment. This implementation aims to provide a more accessible and

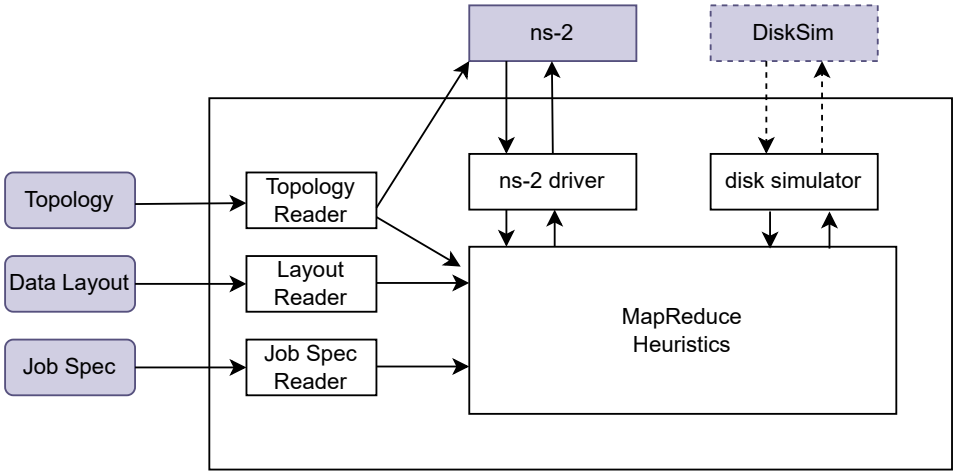


Figure 3.2: Mperf Structure [164]

cheaper way to validate MR operations.

Architecture: In order to implement MR in CloudSim, the MR model was implemented outside of the original CloudSim code to minimize unpredictable behaviour. This simulator provides a simplistic, single-state Map and Reduce computation. Map class gets cloudlets and divides them into a number of MapReduceFiles. MapReduceFile class defines cloudlet with the additional characteristics (i.e., key, map/reduce maker, and pair parent pointer) as input for MapReduce. Finally, Reduce class sorts the marked MapReduceFiles according to their keys.

Analysis: It lacks sufficient support for network link modelling [29, 79]. Also, to provide a bridge between the two implementations, native CloudSim code was modified [79]. MR-CloudSim does not utilize any typical MR Implementation for experimentation. It cannot process streaming media interface with IoT-based applications, job and task scheduling algorithms, data replication and replay of production cluster workloads.

My *MaReClass* framework has shown that the MR-CloudSim is not capable of supporting most MR features. Therefore, *MaReClass* does not recommend MR-CloudSim for MR research.

3.3.1.3 Mumak

The goal of Mumak is to build a discrete event simulator for conditions when a MRH scheduler performs actions on a large-scale workloads.

Architecture: Mumak comprises the following entities: client, job tracker, a task tracker, and simulated engine [114]. As input, Mumak takes a workload and simulates them in a matter of hours, if not minutes, on very few

machines. Additionally, this simulator utilizes data from real experiments to estimate performance metrics (e.g., completion time) for Map and Reduce tasks with different scheduling algorithms [180].

Analysis: Mumak simplifies task execution without modelling the shuffle and sort phase. It omits resource utilisation, such as memory, CPU, and network utilisation. Mumak can use the built-in MR schedulers as-is without any changes. Mumak replays MR workload traces collected with a log processing tool, called Rumen. It then reproduces all conditions of a production cluster. The job submission, inter-arrival, dependencies, task completion time are obtained by Rumen.

However, Mumak is not prepared for IoT-based applications, stream processing, and practical parameter tuning. In cases where data from real experiments does not exist, Mumak cannot estimate the completion time for Map and Reduce tasks. The shuffle and sort phases are not modeled [163].

The *MaReClass* framework has shown that Mumak can perform MR simulations. Therefore, *MaReClass* recommends Mumak for MR research.

3.3.1.4 SimMR

The main goal of developing the MR simulator, called SimMR, was to design an accurate and fast simulation environment [159]. It was also developed for evaluating workload management.

Architecture: SimMR consists of the following three main components: (*i*) trace generator, (*ii*) simulator engine and (*iii*) scheduling policy. These components in its infrastructure implementation work to ensure the allocation of resources for optimization decisions in MR environments. SimMR has scheduling policy that dictates the scheduler decisions on job ordering. The policy also dictates the amount of resources to be allocated to different jobs over time [93]. SimMR employs its simulator engine to engage in network packet-level simulations but not as detailed as MrPerf [164].

Analysis: SimMR allows the tuning up of parameters like schedulers and job queues. SimMR utilizes a propriety engine for its experimentation and does not use any typical MapReduce Implementation. It simulates the MR computing model and can assess various what-if questions and help avoid error-prone decisions. However, SimMR cannot process IoT-based applications nor engage in stream processing.

The *MaReClass* framework has shown SimMR can handle data-intensive MapReduce applications and replay collected job execution traces. Therefore, *MaReClass* recommends SimMR for MR research.

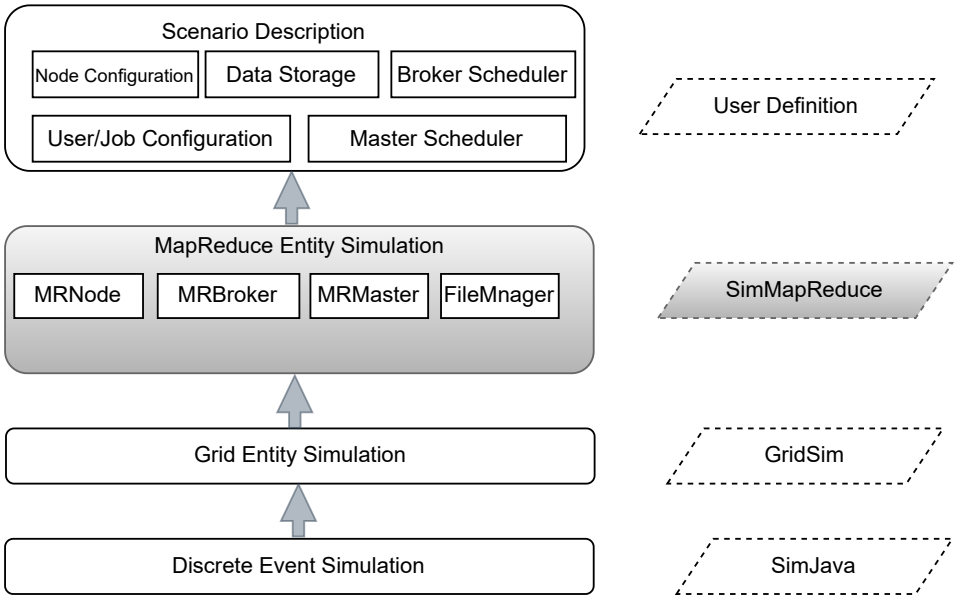


Figure 3.3: SimMapReduce Multi-layer Archtecture [152]

3.3.1.5 SimMapReduce

SimMapReduce(SMR) was designed to evaluate the performance of MapReduce applications under different scenarios.

*Architecture:*The system design of SMR follows a multi-layered architecture. It considers some features such as data locality and dependence between Map and Reduce. These features made the simulator a flexible toolkit and convenient for extension [152]. The main components are: (i) FileManager (ii) MRMaster (iii) MRMasterScheduler (iv) MRNode (v) MRBroker as seen in figure 3.3. These components work to ensure the recording of details of service demands including arrival time, deadline of tasks, locations and size of files. Also, they handle task scheduling algorithms executions, and selection and tuning of key parameters such as input data, speed and reserved MR slot numbers. Scheduling decisions including OS, master and broker schedulings are tackled harmoniously as compared to MrPerf [165].

Analysis: SMRis uniquely capable of simulating various configurations of clusters of shared-memory machines. It can also simulate parallel supercomputers or an extensive collection of networked computers. This simulator was built on SimGrid as its infrastructure Implementations. It utilises a group of java classes for its simulations.

However, it lacks the capability to interface with IoT-based applications, replay the production cluster workload, and execute database traces. SMR

does not support the processing of streaming data.

The *MaReClass* framework has shown that this simulator can perform most MR simulations. Therefore, *MaReClass* recommends SimMapReduce for MR research.

3.3.1.6 IoTSim

This simulator is built on top of CloudSim. IoTSim allows the simulation of IoT applications and inherently supports big data processing. This simulator inherently supports big data processing such as MR to facilitate research and the analysis of IoT-Based applications. It analyses the impact and performance of IoT-based applications [154, 180].

Architecture: IoTSim has five main layers. These are the CloudSim core simulation engine, CloudSim simulation layer, storage layer, extensive data processing layer, and the user code layer [93].

Analysis: IotSim permits the modelling and simulation of network usage, storage and the processing of virtual machines. IoTSim supports the data transfer via packet-level simulations. The simulator allows the tuning of parameters to ensure the scalability of resources. IoTSim support resource allocation by modelling large scale multiple IoT applications to run simultaneously in shared cloud data centres compared to MrPerf [165] and MR-CloudSim [79]. IoTSim supports IoT-based big data processing using the MapReduce model.

However, IoTSim does not utilize any typical MapReduce Implementation. IoTSim does not replay the production cluster workload, and execute database traces.

My *MaReClass* framework has shown IoTSim is the only MR Simulator capable of interfacing with IoT applications. Therefore, my framework recommends IoTSim for MR research.

There are a couple offshoots from IoTSim which are used for various applications. IoTSim-Stream [16] is utilised for modelling stream applications in cloud simulation. IoTSim-Edge [77] is used for modeling the behavior of Internet of Things and edge computing environments. Iotsim-sdwan [7] is used for interconnecting distributed datacenters over software-defined wide area network (sd-wan). IoTSim-Osmosis [44] is used for modelling & simulating IoT applications over an edge-cloud continuum. Lastly, IoTSim-Osmosis-RES [149] is used for autonomic renewable energy-aware osmotic computing.

3.3.1.7 HSim and HaSim

The HSim or HaSim simulator is designed for MRH applications. The contributions of this simulator lie in its high efficiency to simulate the dynamic behaviours of Hadoop environments.

Architecture: Many Hadoop parameters can be modelled in the simulator by tuning their values [85, 103]. The critical parameters for running this simulator include node-, cluster-, Hadoop system- and HSim parameters [17]. Also, it can be used to study the scalability of MR applications which might involve several of nodes [172]. HSim/HaSim is suitable for heterogeneous computing environments.

Analysis: HSim allows the computing of nodes with varied processing capabilities to parallelly utilize MapReduce applications. It allows packet-level simulations through the transfer of tasks to node. Due to its Hadoop infrastructure implementation, it managements workloads and allocation resources via several data nodes. HSim utilizes its two components (MapperSim and ReducerSim) to write, copy and update files unto various data nodes for task processing and trace executions. However, HSim does not support the processing of streaming data and interfacing with IoT-based applications.

My *MaReClass* framework has shown that HSim is efficient and capable of simulating MRH clusters as compared to MPerf [165]. Therefore, *MaReClass* recommends this simulator for MR research due to the number of features it supports.

3.3.1.8 WaxElephant

WaxElephant is designed to help Hadoop operators in scalability analysis and parameter tuning.

Architecture: The simulator comprises a pluggable job scheduling module, a load generator, and a simulator engine [129]as seen in figure 3.4. The simulator's job scheduling module receives jobs from the load generator and monitors the status of computing nodes. The load generator is responsible for producing MR jobs. It replays jobs collected from a historical log or synthesizes a set of jobs that follow user-defined statistical properties [65,166].

Analysis: This simulator is capable of simulating the job execution phases of Hadoop clusters compared to Mumak [114]. WaxElephant simulates the complete progress process of map and reduce tasks as compared to MrPerf [165]. Also, WaxElephant ensures parameter tuning required in every efficient MR simulator. Some parameters utilised for simulating the cluster environment include the cluster topology and the network transfer speed. The network transfer speed allows the definitions of the various parameters

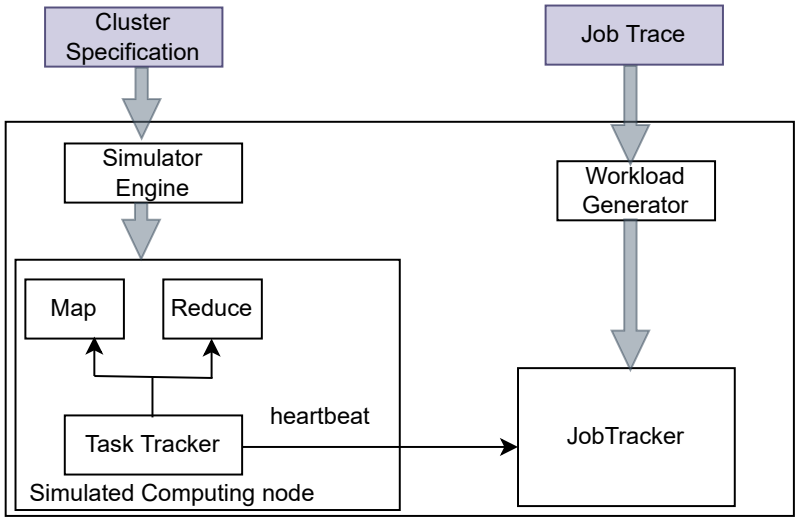


Figure 3.4: Architecture of WaxElephant [129]

required for the packet-level simulations. However, WaxElephant is incapable of stream processing and interfacing with IoT-based applications.

WaxElephant has most of the features required to run MapReduce simulations according to my *MaReClass* framework. *MaReClass* recommends WaxElephant for most MRH research.

3.3.1.9 MRSim

The goal of the MRSim project was to design and implement a MapReduce simulator to model the behaviour of data mining algorithms in hadoop environments.

Architecture: MRSim was created by extending the discrete event engine used by SimJava to simulate the Hadoop environment [173]. MRSim utilises its network topology component to initiate the connection between nodes and for task transfers. The System is designed using object oriented based models. The CPU, HDD and Network Interface models were designed to be the basic blocks which can be grouped in PC machine entity as shown in figure 3.5.Moreover, the network topology is responsible for monitoring job completion times as part of the packet-level simulations. MRSim simulates data splits locations and splits replications in local racks only. In contrast to MPerf, MRSim simulates much more details for shared multi-core CPUs. MRSim considers memory buffers, merge parameters, parallel copy and sort parameters in its cluster configurations. The tuning of these parameters ensures accurate predictions of MR based applications. This simulator enables

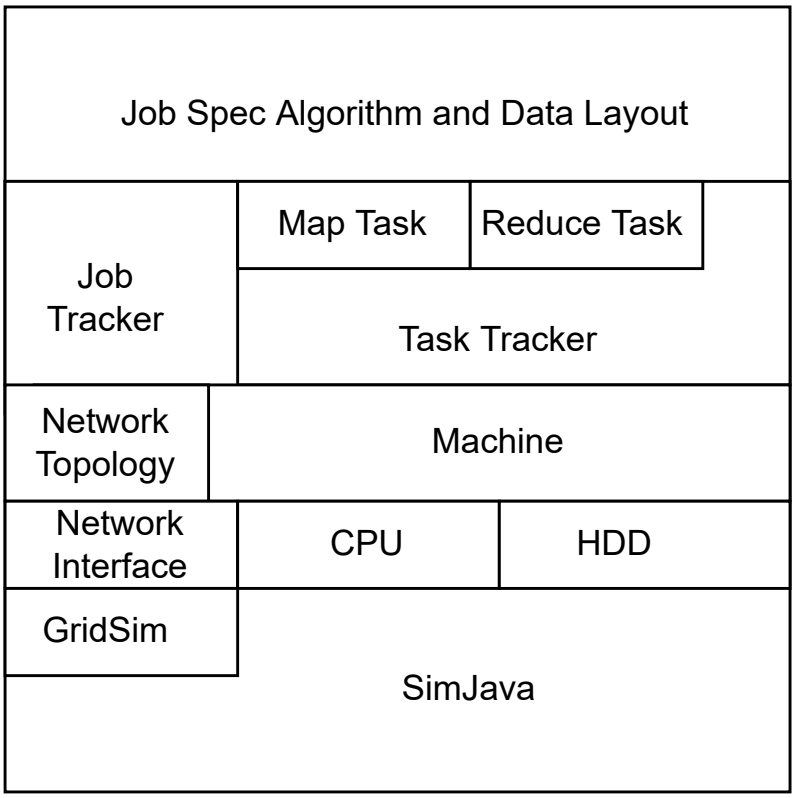


Figure 3.5: MRSim Structure [173]

users to estimate the best job configuration to get optimum performance for certain algorithms [128].

Analysis: MRSim offers functionalities for measuring the scalability of MR applications. Also, it studies the effects of various Hadoop setup configurations on the behaviours of these applications during job processing. MRSim analyses the behaviours of Hadoop job completion times and hardware utilization which are related to the Hadoop infrastructure implementation [71] as compared to MrPerf [165].

However, MRSim cannot interface with IoT-based applications and replay production cluster workloads. No interface is provided to modify the framework algorithms such as task scheduling and data distribution. Also, MRSim cannot process streaming data.

MRSim has most of the features required for MR simulation according to its assessment by my *MaReClass* framework. Therefore, *MaReClass* recommends MRSim for MR research.

Table 3.9: A Summary of Limitations

Simulators	Limitations
MrPerf [164]	(1) It does not simulate vital aspects of the Hadoop platform, such as speculative execution. (2) It cannot handle reservation and scheduling schemes. (3) It does not support the processing of streaming data. (4) It does not interface with IoT-based applications.
MRSim [71]	(1) No interface is provided to modify the framework algorithms such as task scheduling and data distribution. (2) It cannot interface with IoT-based applications. (3) It cannot replay production cluster workloads.
SimMR [159]	(1) It does not process streaming data and interface with IoT-based Applications.

3.3.1.10 MRSG

The simulator aims to facilitate research on the behaviour of MR platforms and possible technological changes.

Architecture: MRSG is developed on top of SimGrid, a simulation framework for evaluating cluster, grid, and P2P (peer-to-peer) algorithms and heuristics [128]. This simulator allows users to define task costs and intermediary data [30]. MRSG provides a complete API to translate theoretical algorithms, such as task scheduling and data distribution, into executable code. Also, it provides the ease of changing and testing different cluster configurations [89]. MRSG reproduces the real execution of MR accurately, even with different cluster configurations. It utilises the simgrid APIs to initiate task transfers between nodes on various clusters.

Analysis: MRSG provides a complete API to translate theoretical algorithms, such as task scheduling and data distribution, into executable code. The APIs allows users to configure the methods of objects to achieve specific resource scaling targets. MRSG allows data replication, to enable or disable the speculative execution as compared to MrPerf [165]. MRSG allows the modification of MR mechanisms as compared to MRSim [71]. However, this simulator cannot process streaming data and interface with IoT-based Applications. It does not depend on any MapReduce Implementation for experimentation.

My *MaReClass* framework has shown that the MRSG is capable of simulating MR effectively.Therefore, *MaReClass* recommends MRSG for MR research.

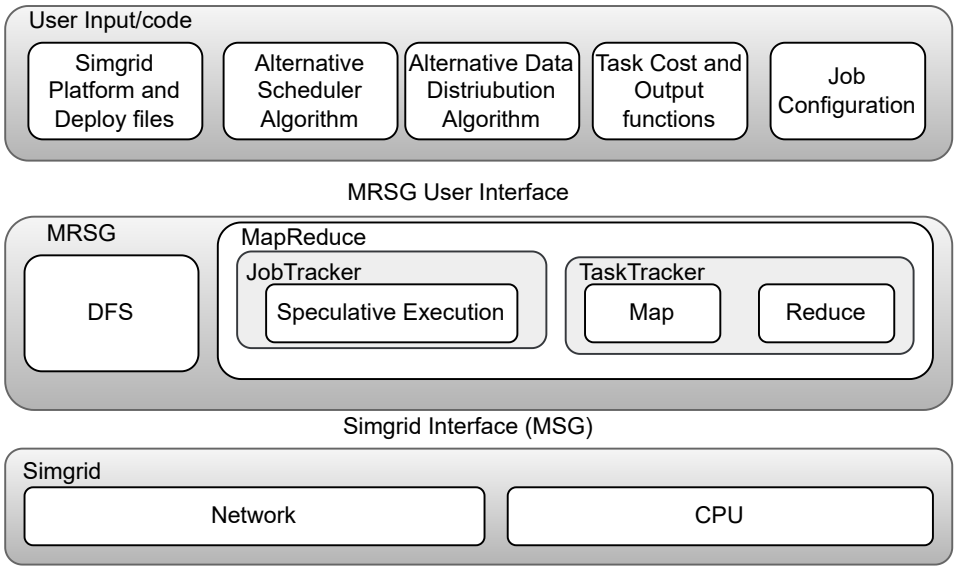


Figure 3.6: MRSG Archtecture [89]

3.3.1.11 BigDataSDNSim

This simulator enables the modeling and simulation of the big data management system YARN, its related programming models MR, and SDN-enabled networks in a cloud computing environment.

Architecture: BigDataSDNSim (BDSS) is categorised into two main layers: programming and infrastructure and big data. This simulator offers the holistic modelling and integration of MR Big Data Management System-based models that are compatible with Software Defined Network(SDN) network functions in cloud infrastructures. BigDataSDNSim (BDSS) provides an infrastructure for researchers to quantify the performance impacts of MR

Table 3.10: A Summary of Limitations

Simulators	Limitations
SimMapReduce [152]	(1) System functionalities such as storage topologies and friendlier GUI are absent. Also, redundant execution for handing machines features and data loss are unresolved.
MR-Cloudsim [79]	(1) It lacks sufficient support for network link modelling. (2) It cannot represent real world systems.
WaxElephant [129]	(1) It does not support the processing of streaming data. (2) It does not interface with IoT-based applications.

Table 3.11: A Summary of Limitations Cont'd

Simulators	Limitations
Mumak [114]	(1) The shuffle and sort phases are not modelled. (2) It cannot estimate the completion time for Map and Reduce tasks in the absence of real experiments.
Hasim [85]	(1) It does not support the processing of streaming data. (2) It does not interface with IoT-based applications.
MRSG [89]	(1) It does not have the ability to handle faults and volatile environment. (2) It cannot process streaming data nor interface with IoT-based Applications.

applications in terms of a joint-design of host and network [6]. BDSS contains a variety of application-network policies for diverse purposes (e.g., scheduling and routing), which can be seamlessly extended without a deep understanding of the complex interactions among BigDataSDNSim's components. BDSS supports network packet-level as it closely monitors the bandwidth, transmission-, processing-, and total completion times of tasks during data processing.

In order to ensure that the design goals were met; key policies were modelled and implemented. They are; MR application selection , HDFS replica placement, VM-CPU scheduling, Traffic and routing algorithms policies. These policies ensure the support of BDSS for some of the primary features required in an MR simulator.

Analysis: BDSS utilises the application selection policy to determine the selection criteria based on given QoS. This policy meets the ACM feature expected in MR simulators. The HDFS replica placement policy can be extend to include general policies including scheduling and VM usage. The features widely supported include tasks scheduling, data replication and the MR computing model. However, this simulator cannot process streaming data and interface with IoT-based Applications.

My *MaReClass* framework has shown that the BDSS is capable of simulating MR experiments related to SDNs effectively. Therefore *MaReClass* recommends BDSS for MR research.

3.3.1.12 HDMSG-EXTENSION

This simulator was designed as tool that supports MR job execution phases on large scale workloads. The simulator is group of classes that supports big

Table 3.12: A Summary of Limitations Cont'd

Simulators	Limitations
HSim [103]	(1) It cannot validate the accuracy of simulating the behaviors of a larger scale Hadoop cluster. (2) It does not interface with IoT-based applications.
IoTSim [180]	(1) It does not replay the production cluster workload and execute database traces.
BigDataSDNSim [6]	(1) It does not model and simulate of stream paradigms in the context of big data Software Defined Network powered cloud environments. (2) It does not interface with IoT-based Applications.

data processing with Simgrid as the main backbone (available on GitHub [1]). HDMSG-EXTENSION (HDEXT) simulator permits the configuration of key parameters to foster the scalability of workloads and efficient allocation of resources. The parameters includes the number of nodes, CPU cores, bandwidth, latency metrics, and the nodes' speed. Other parameters that can be defined include the number of mappers and reducers, file input size (in megabytes), and block size (HDFS chunk size in megabytes).

HDEXT supports the configuration of network specific parameters to ensure efficient packet-level simulations. HDEXT is capable of simulating various configurations of MR simulation, while replicating synchronize data with zero data loss. The simulator supports the simulation of speculative execution compared to MrPerf [165]. Also, It allows the modification of MR mechanisms as compared to MRSim [71]. The simulator utilise Simgrids APIs to configured the methods of objects to translate algorithms such as snapshot capturing and task monitoring into executable code. Also, HDEXT applies Simgrids APIs to define task costs and intermediary data for various data processing functions. The task cost allows the definition of simulation start and completion times. However, the simulator cannot process streaming data. Also, it cannot interface with IoT-based applications and replay production cluster workloads.

My *MaReClass* framework has shown that the HDEXT is capable of simulating MR effectively.Therefore, *MaReClass* recommends HDEXT for MR research.

[1] `https://github.com/EbenezerKomlaGavua/MapReduce_Snapshots`

Table 3.13: Yearly Improvements of MR Simulators

Simulators	Year	Improvements
Mumak [114], MPerf [164]	2009	(1) Provides a discrete event simulator for Mapreduce scheduler data processing on a large-scale workload. (2) Focuses on a realistic phase-level simulator towards designing, provisioning, and fine-tuning of Hadoop setups.
MRSim [71]	2010	(1) Captures the effects of different configurations of MR applications. Models Hadoop setup's of an algorithm's behavior in terms of speed and hardware utilization.
SimMapReduce [152], SimMR [159]	2011	(1) Enhances the modelling of multi-layer scheduling algorithms on user-level, job-level or task-level by extending preserved classes. (2) Supports fast simulation of complex multi-hour workload in less than a second.
MR-Cloudsim [79], WaxElephant [129]	2012	(1) Provide an easier way to examine MR model in a data center. (2) Synthesizes workloads and executes them based on statistical characteristics of workloads.

3.3.1.13 DISSECT-CF

DISSECT-CF (as discussed in 2.4) supports data replication and locality and IoT-based applications [84, 109]. It replays execution traces and processes data intensive applications. However, it did does not support the MapReduce computing model and stream processing.

My *MaReClass* framework has shown DISSECT-CF is efficient and capable of simulating most cloud computing experiments. Therefore *MaReClass* recommends DISSECT-CF for research.

Table 3.14: Yearly Improvements of MR Simulators Cont'd

Simulators	Year	Improvements
HSim [103], Hasim [85], MRSG [89]	2013	(1) Present accurate simulation of the dynamic behaviors of Hadoop clusters. (2) Investigates the impacts of the large number of Hadoop parameters by tuning their values. (3) Supports the study of the scalability of MR applications which might involve hundreds of nodes. (4) Provides functionalities absent in other simulators, such as the speculative execution mechanism and data replication. (5) Allows the definition of task costs and intermediary data through API functions, rather then a single value.
IoTSim [180]	2017	(1) Offers support for the simulation of IoT applications.
BigDataSDNSim [6]	2021	(1) Provides support for simulating and evaluating the performance of big data applications in Software Defined Network enabled cloud data centers.

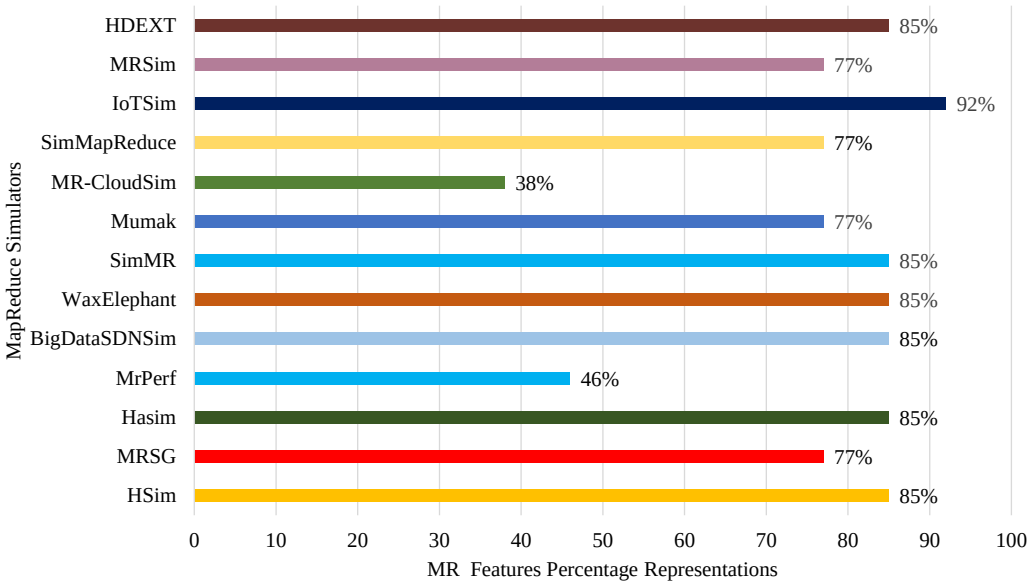


Figure 3.7: Percentage Representation of MR Simulators Support for Specific Features

3.3.2 Evaluation of MapReduce Simulators

My *MaReClass* framework has shown the strengths and weaknesses of the MR simulators as summarized in tables 3.6 to 3.12. A summarized assessment of the yearly improvements provided by the MR simulators showing efforts by researchers is shown in tables 3.13 to 3.14

A technical analysis summarized in table 3.4 to 3.5 shows specific details associated with the features already discussed. These details show the specific programming languages utilised in creating the simulators, the platforms used on which certain simulators were developed and mostly especially the last updated date of the MR simulators. The last updated date shows how current a particular simulator is and why a researcher might opt to use it for an experiment. Other features shown include the availability of the simulator and its associated documentation and any existing energy model. Such features are required by researchers in order for them to make informed decision about the choice of MR simulators for research.

Each of the the MR simulators analysed support the feature discussed to some extent. Therefore, equation 3.1 was utilised to determine the percentage representation of the support for the MR features as shown in figure 3.7. However, in the situation where there is a partial support for a specific feature, equation 3.2 is applied to determine the percentage partial support.

$$PM_{SS} = (\frac{M_{SS}}{TNM_{SS}}) * 100 \tag{3.1}$$

$$PM_{SPS} = (\frac{M_{SS} - B}{TNM_{SS}}) * 100 \tag{3.2}$$

Where PM_{SS}, M_{SS} , TNM_{SS} are the positive rational numbers for Percentage MR Simulators Feature Support, MR Simulators Feature Support and Total Number of MR Simulators Feature Support. B is the positive rational number quantity of feature supported by a MR Simulator. PM_{SPS} is the the positive rational number for MapReduce Simulators Feature Partial Support.

Figure 3.7 shows that aside MR-Cloudsim [79] and MrPerf [164], all the other simulators support most of the features required for conducting MR research with percentage representation above 50%. This makes most of the simulators analysed recommendable for research and development as seen in table 3.15.

Table 3.15: Recommendation for MR Research

Simulators	Recommended	Not Recommended
MrPerf [164]	–	✓
MRSim [71]	✓	–
SimMR [159]	✓	–
SimMapreduce [152]	✓	–
MR-Cloudsim [79]	–	✓
Waxelephant [129]	✓	–
Mumak [114]	✓	–
Hasim [85]	✓	–
MRSG [89]	✓	–
HSim [103]	✓	–
IOTSim [180]	✓	–
BigDataSDNSim [6]	✓	–
HDMSG-EXT [64]	✓	–

3.3.3 Recommendations

The systematic literature reviewed carried out in this chapter has highlighted vital features required in cloud simulators for MR research. Therefore, it is recommended that simulator developers focus on incorporating some of these unavailable features into their works. Simulator developers are encouraged to focus on notable features such as data locality and dependence between Map and Reduce, as they provide essential services. Simulator developers should focus on parameter tuning, scalability analysis, and job/task scheduling algorithms and decisions. These features are the distinguishing factors of the MR programming model. Therefore, neglecting them when developing MR simulators will generate inaccurate results.

The researchers interested in monitoring speculative execution mechanisms and data replication should select frameworks that can simulate cluster environments. Simulating IoT applications is gradually becoming an important area for researchers. Hence, simulators with IoT capabilities are recommended for such research work since they have the ideal platform for IoT-based applications. Most MR researchers prefer simulators that can replay production cluster workloads. Therefore, simulators that replay production cluster workloads with different scenarios of interest, assessing various what-if questions and helping to avoid error-prone decisions, are recommended.

It is recommended for researchers interested in specific simulators, to

evaluate the frameworks via *MaReClass*. Since it highlights the primary features supported by MR simulators.

3.4 Summary

This chapter focused on the design and implementation of a MR Classification framework(*MaReClass*). This was done via systematic literature review.

First, Thirty cloud computing simulator features were identified including network topology support and job completion time estimation. Second, a MR-specific evaluation refined the potential criteria to the thirteen themes most relevant to Hadoop/MR simulations.

Third, MR simulators were identified through literature review. The thirteen MR themes were utilised as evaluation criteria to assess the MR simulators to determine, which specific MR experiments they were suitable for. The evaluation work showed that the absence of any of the thirteen criteria rendered detailed MR simulations incomplete.

Recommendations have been made for what simulator developers and researchers should consider when planning their MR research.

Chapter 4
Design and Implementation of a Speculative Execution Strategy on MapReduce Hadoop

4.1 Introduction

This chapter discusses my solution that tackles speculative execution on MapReduce Hadoop (MRH). The chapter begins with a background discussion on MapReduce Hadoop to lay the foundation for my approach. In the next section, we will discuss four algorithms that constitutes my solution. The evaluation section discusses the implementation of the algorithms and the results of the experiments carried out.

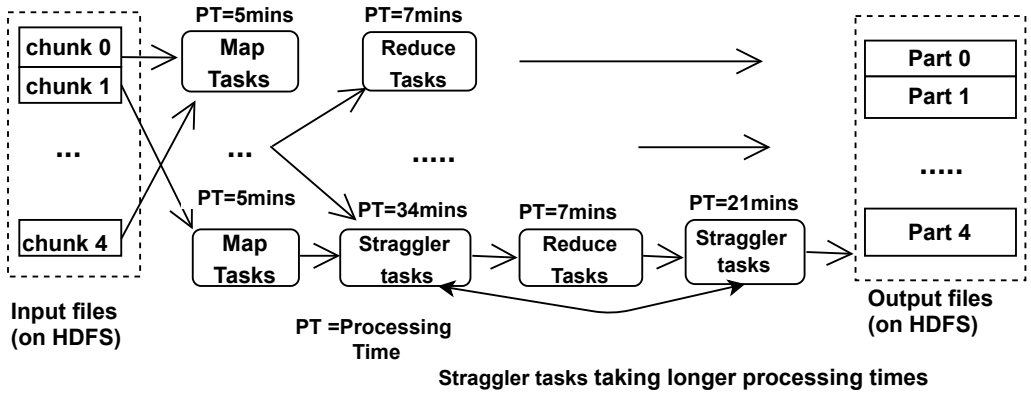


Figure 4.1: Speculative Execution

4.1.1 Background

4.1.1.1 Speculative Execution

The MRH model brings an essential benefit by automatically processing faults. When a node is at fault or task is relative slow, MRH randomly chooses another node to execute the straggler tasks. These faults are mainly due to IO contentions, background services, hardware behaviours, unbalanced load or uneven distribution of resources and other reasons [132]. These faults result in inconsistent speed between multiple tasks running the same job. Without this mechanism of speculative execution, a job would be as slow as the misbehaving straggler task. Some straggler tasks run significantly slower than other tasks as shown in figure 4.1, where the straggler tasks take more processing times than the normal MR tasks. This process is Hadoop's speculative execution [170].

4.2 Design of Speculative Execution Strategy

This section focuses on the design of our *Haspeck* Solution. *Haspeck* is designed to improve job performance on MRH. The approach consists of three algorithms that are interconnected to ensure correct determination of task run times, appropriate selection of backup tasks and reduction in the consumption of system resources. The goals of this section are:

- To design an algorithm that captures task run times during data processing on mappers and reducers. This is achieved by repetitive capturing of the task run times at specific intervals.
- To design an algorithm that monitors task performance on their nodes. The purpose of this algorithm is to ensure that, the straggler tasks are rescheduled to available nodes and processed to reduce high consumption of resources.
- To develop an algorithm that monitors tasks instances to ascertain the completion of uploaded jobs. This algorithm is implemented with the previous ones to ensure that all jobs uploaded are completely processed.
- To implement K-means clustering algorithm to determine straggler tasks. The K-means clustering algorithm is applied with the Silhouette Coefficient to validate the outputs of the clustered data sets.

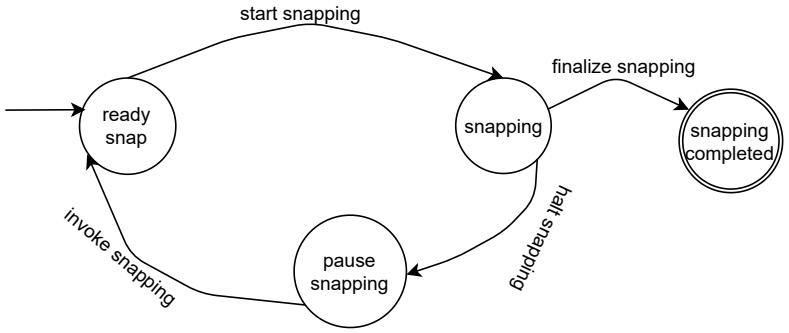


Figure 4.2: Snapshot Capturing State Diagram

- To assess the algorithms on scalable configurations of MRH to prove their applicability. A survey of industry and real-life MRH configurations is conducted to ensure that the solution is applicable in real life and industry.

4.2.1 *Haspeck* Solution on MapReduce

Our approach comprises of snapshot capturing, task performance monitoring, and task instance monitoring algorithms as seen in algorithms 1 to 3, and figures 4.2 and 4.4. Figure 4.2 shows the snapshots capturing state transitions during task processing. The algorithms works together to ensure that straggler tasks are detected correctly and processed as backup tasks. This approach is designed to dynamically collects real-time data from all types of environments. This makes our approach more usable as compared to a few existing approaches which struggle in heterogeneous environments. The details of the algorithms are discussed below.

Snapshots Capturing Algorithm This algorithm initialises with task processing to foster the capturing of task run times as seen in figure 4.4. Nodes (N) are monitored before task processing begins. This is done to capture the commencement of task processing (*start times*), as seen in the snapshot capturing state diagram in figure 4.2. When job processing commences, data is uploaded into the system for task processing to commence.

The *SnapStateFunction* is activated, which causes *Snap_state* (C_s) to be updated to *ready_snap* as seen in lines 1 to 3 of algorithm 1. However, if a task is not ready (due to a fault), C_s is updated to *pause snapping* and the system is checked (for the task to be restarted) as seen in lines lines 4 to 7

Algorithm 1 *Snapshot Capturing Algorithm*

Require: Variables: $T_s = Task\_state, C_s = Snap\_state,\ N = Node$
Require: Variables: $Q_I = Tasks\_Instances,\ i = counter\ for\ Q_I$
Require: Variables: $G_s = captured\_snapshots$
Require: SnapStateFunction: $SnapState : C_s \rightarrow \{snap\_ready, snapping, snap\_paused, snap\_completed\}$

1: **for** $i < Q_I$ **do**
2: **if** $T_s = ready$ **then**
3: $C_s := snap\_ready$
4: **else**
5: $C_s := pause\_snapping$
6: *check the system and restart the task*
7: **end if**
8: **while** $T_s = running$ **do**
9: $C_s := snapping$
10: *Save the captured snapshots*
11: $G_s = G_s + 1 snapping$
12: **end while**
13: $i = i + 1$
14: **end for**
15: **if** $T_s = terminated$ **then**
16: $C_s := pause\_snapping$
17: **end if**
18: **while** $i <= Q_I$ **do**
19: $C_s := snapping$
20: *Save the captured snapshots*
21: **if** $T_s = completed$ **then**
22: $C_s := snapping\_completed$
23: *Save the captured snapshots*
24: $G_s = G_s + 1 snapping$
25: **end if**
26: $i = i + 1$
27: **end while**

While the updated data is being processed, (C_s) is updated from *snap ready* to *start_snapping* as as seen in lines 8 to 13. Task run times are captured and saved as seen lines 9 to 10. Task run times are captured on all nodes and saved on snapshots text files. Snapshots are captured repeatedly for all the tasks running on nodes. The accumulated local snapshots captured constitute the global snapshots (G_s).

The details captured include the start time, completion time, the node on which the task is running, task identification, and task status. The start and completion times are captured to monitor the rate of task processing. The task status provides the task performance information.

Additionally, the *TaskStateFunction* is activated, which causes *task state* (T_s) to be updated from *ready* to *running* as seen in the task state diagram in figure 4.3. T_s remains unchanged until all the tasks are completely processed. Then, it transitions from *running* to *completed*. However, when a task's run time is unnecessarily longer that expected, the task is suspended causes T_s to be updated to *terminated* and C_s to *pause snapping* as seen in lines 15 to 17.

When a configurable percentage of the tasks have been processed with captured run times; K-means clustering algorithm is employed to classify the captured data on the snapshots text files. The clustering is utilised to determine the straggler tasks. The straggler tasks identified from the clustered data are then processed as backup task on available nodes. This causes T_s to be updated from *terminated* to *rescheduled* as seen in figure 4.3.

When the tasks rescheduling is completed, the re-processing of the backup tasks processing commences. This causes T_s to transition from *rescheduled* to *ready*. The backup tasks are processed together with the snapshot capturing until all the tasks are completely processed as seen in lines 18 to 27.

Task Performance Monitoring Algorithm The task performance monitoring and the snapshot capturing algorithms work concurrently to ensure job performance improvement, as seen in figure 4.4.

Algorithm 2 is applied during task processing to monitor and evaluate task performance. When data processing begins, all tasks (i.e., $t = t_{1,2,3,...n}$) are expected to process data at the same rate. These tasks are allocated processes on compute nodes as seen in lines 1 to 2.

The *TaskStateFunction* is activated which causes T_s to be updated to *ready* as seen in line 5. Tasks-instances (I_t, where $t \in I_t$) are monitored

Algorithm 2 *Task Performance Monitoring Algorithm*

Require: Variables: $Q_I = Tasks\_Instances, T_R = runningtasks$
Require: Variables: $N_i = Node,\ A_n = AvailNodes, T_C = completedtasks$
Require: Variables: $T = \{t_1, t_2, t_3, ...t_n\},\ T_s = Task\_State$
Require: Variables: $T_{ET} = task\ execution\ time, T_{MET} = task\ maximum\ execution\ time$
Require: TaskStateFunction: $TaskState : T_s \rightarrow \{ready, running, termination, reschedule, completion\}$

Begin Tasks Processing in the Map or Reduce Phase
for $t <= T$ **do**
  $status \leftarrow checkTasksStatus$
  switch $(status)$
  $T_s := ready$
  case *still_Running*:
    monitor the progress of the task
    $T_s := running$
    if $T_{ET} > T_{MET}$ **then**
      terminate the task
      $T_s := terminated$
      reschedule straggler tasks on available nodes
      $T_s := rescheduled$
      $T_{\mathrm{R}} \leftarrow (T_{\mathrm{R}} + t)$
    else
      process all the tasks
    end if
  case *finished_Running*:
    Tasks completely processed
    $T_s := completed$
    Output results
    $A_{\mathrm{n}} \leftarrow (A_{\mathrm{n}} + N_i)$
    $T_C \leftarrow (T_{\mathrm{C}} + t)$
    $Q_i \leftarrow (Q_{\mathrm{i}} - t)$
  end switch
end for

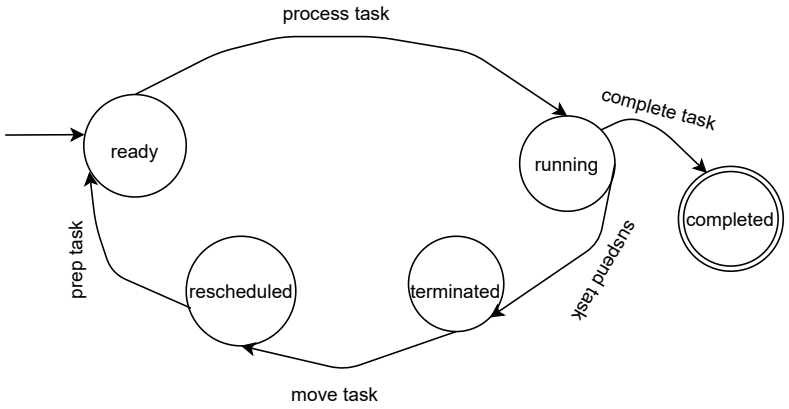


Figure 4.3: Rescheduling of Tasks State Diagram

to determine whether they are still running (T_R) or are completely processed (T_c) as seen in lines 6 and 18.

During task processing, tasks which have relatively longer running times than the maximum execution times (T_{MET}) of the tasks being processed are terminated as seen in lines 6 to 10. This causes T_s to be updated to *terminated* as seen in line 11 and figure 4.3. The K-means algorithm is applied to cluster all the captured task run times as seen in figure 4.5. The tasks identified as straggler tasks are rescheduled as seen in lines 12 to 14. T_s transitions from *terminated* through *rescheduled* to *ready* as seen in figure 4.3.

The states of the tasks which do not exhibit relative longer run times, transition from *ready* through *running* to *completed*. This enables their compute nodes to be availed for processing backup tasks, and also reduces the number of task instances. These processes are seen in lines 18 to 26.

Task Instance Monitoring Algorithm A vital aspect of our approach is monitoring task instances (Q_i). The number of tasks are monitored throughout their processing stages as is seen in algorithm 3. When the number of active tasks are exhausted, job processing ends as seen in lines 3 to 4. Otherwise, the task processing continues until the jobs generated are completely processed as seen in lines 5 to 8.

Also, backup task instances are monitored during their processing until they are all completely processed.

We can now discuss the K-mean Clustering Algorithm in the context of our approach.

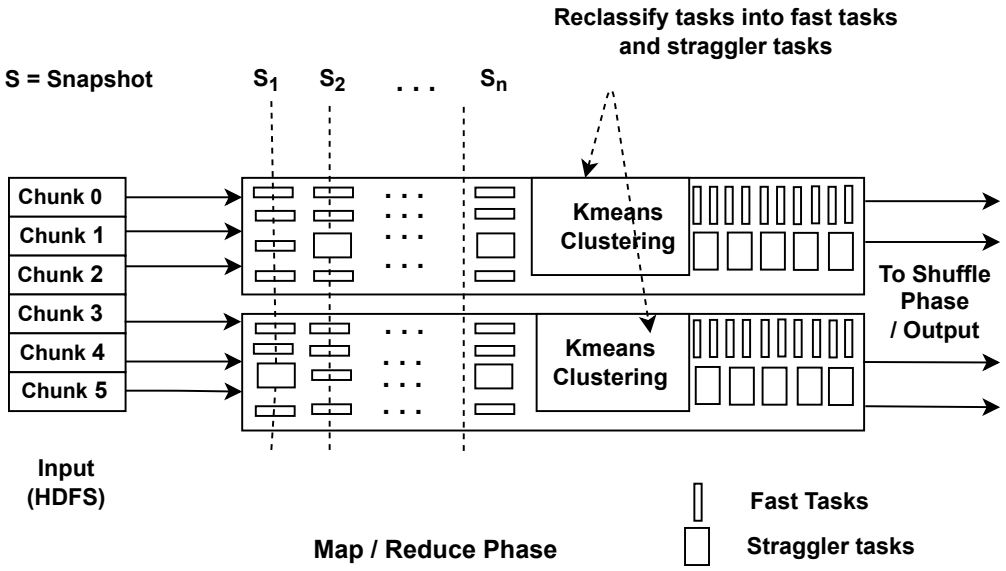


Figure 4.4: Structure of Task Performance Monitoring Algorithm

4.2.2 Identifying Straggler Tasks with K-means Clustering Algorithm

The identification of straggler tasks during job processing is a challenge that required addressing in our approach. This is achieved via the adoption of a clustering technique, after task run times have been captured via snapshots capturing.

Clustering was considered because it is the type of unsupervised machine learning technique which aims at partitioning sets of objects into groups called *clusters*. These *clusters* can be mutually exclusive or they may overlap,

Algorithm 3 *Task Instance Monitoring Algorithm*

Require: **Variables**: $Q_I = Tasks\ Instances$

1: *Check the number of Tasks Instances*
2: **for** $t \in T$ **do**
3: **if** $|Q_i| = 0$ **then**
4: *Stop tasks monitoring*
5: **else**
6: *Continue with tasks monitoring*
7: **end if**
8: **end for**

Table 4.1: KMeans Clustering Silhouette Scores of No-Disrupted Data Clusters

Figure Number	Silhouette Scores
4.6a	0.685
4.6b	0.604
4.6c	0.638
4.6d	0.615
4.7a	0.985
4.7b	0.985
4.7c	0.985
4.7d	0.985

depending on the approach used. It is in contrast to the supervised learning techniques where the goal is to make predictions about output value y given an input object or instance x [40]. This made the choice of clustering suitable for our approach since there was no need for training any data set to achieve our groupings.

Additionally, we considered the K-means clustering as the clustering technique for our approach because it is a hard clustering algorithm. K-means partitions a set of n objects into k clusters, so that the resulting intra-cluster similarity is high but the inter-cluster similarity is low [21]. It was the most suitable clustering algorithm for our approach since two distinct groups are required; thus fast tasks and straggler tasks.

Our approach applied the K-means clustering algorithm to categorise task run times (dataset) received from the snapshot capturing algorithm. The data set saved on snapshots text files during the map or the reduce phases are clustered into fast and straggler tasks as seen in figure 4.5.

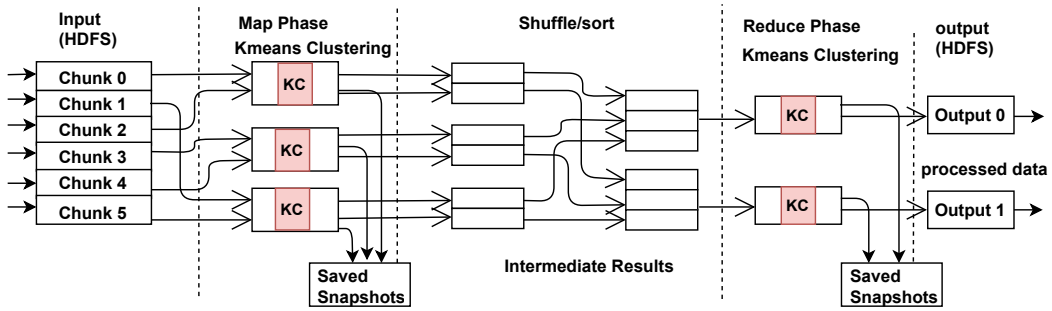
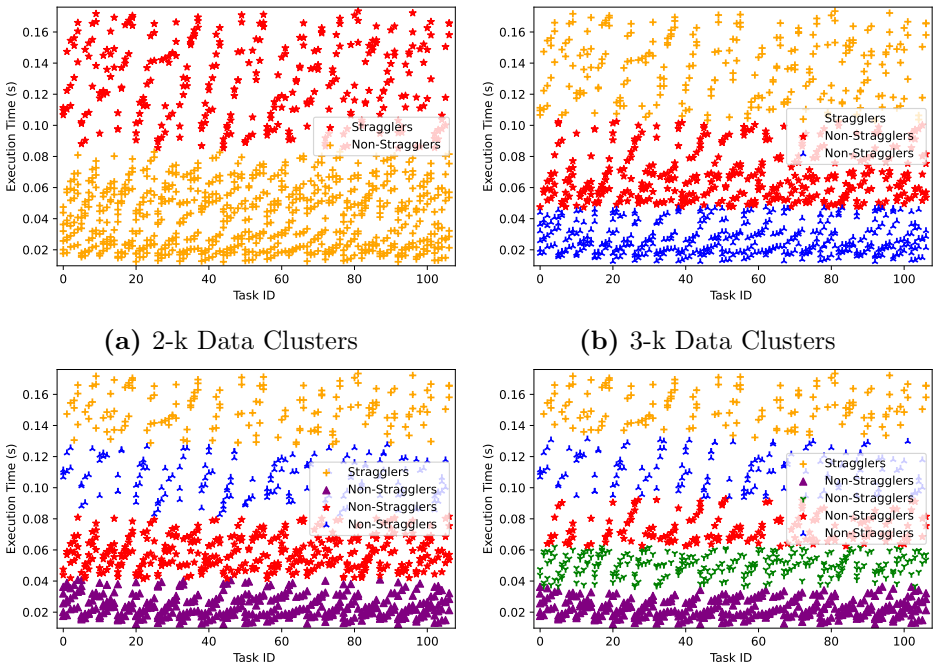


Figure 4.5: Structure of Our Approach Implementation with Kmeans Clustering



(a) 2-k Data Clusters **(b)** 3-k Data Clusters

(c) 4-k Data Clusters **(d)** 5-k Data Clusters

Figure 4.6: Kmeans Data Clusters of Tasks Execution times from 20 Nodes by 8 Cores Data Centre with No-Disruption

K-means optimizes the distance between the task run times to their centre points, as seen in equation 4.1 [177].

$$J(V) = \sum_{i=1}^{C_d} \sum_{j=1}^{C_i} (|x_i - y_j|)^2 \tag{4.1}$$

The attributes of the data set captured for the clustering include; hostname, task id, and partial execution times of the task's progress. K-means utilizes the task id and the execution times to cluster the data into two categories (fast and straggler tasks). The cluster with shorter average of the partial execution times are the fast tasks, while the remainder of the tasks are considered as straggler tasks.

Also, the K-means algorithm is implemented in our work as a decision-making tool. Thus, it directs whether to process backup tasks or not. There are cases where the dataset presented for clustering is uniform. However, K-means still tries to cluster it. Thus, clustering results require validation to determine the goodness of fit of the data clusters created as seen in figures 4.6a to 4.6d.

In order to ensure the effective creation of clusters, four clustering validation techniques were considered. These are Dunn [68], Davie-Bouldin, Calinski-Harabasz indices, and the Silhouette score [157]. However, the silhouette score was selected for our approach. The first three techniques were not implemented because of the following reasons: first, although the Calinski-Harabasz index defines how dense and separated a cluster is, the absence of upper- and lower-bounds ranges made it inapplicable. Second, the Davies-Bouldin index utilizes zero (0) as the upper bound; and values closer to zero indicate a better partition. Moreover, the Davies-Bouldin index did not have a lower bound. In the case of the Dunn index, higher indices indicate better clustering. However, the absence of a lower bound makes it inapplicable in our context. Since, without a closed range of clustering validation values, a deterministic algorithm based on them would be unreliable. Also, the presence of the upper-lower bounds fosters faster approach and makes the choice inappropriate for our research.

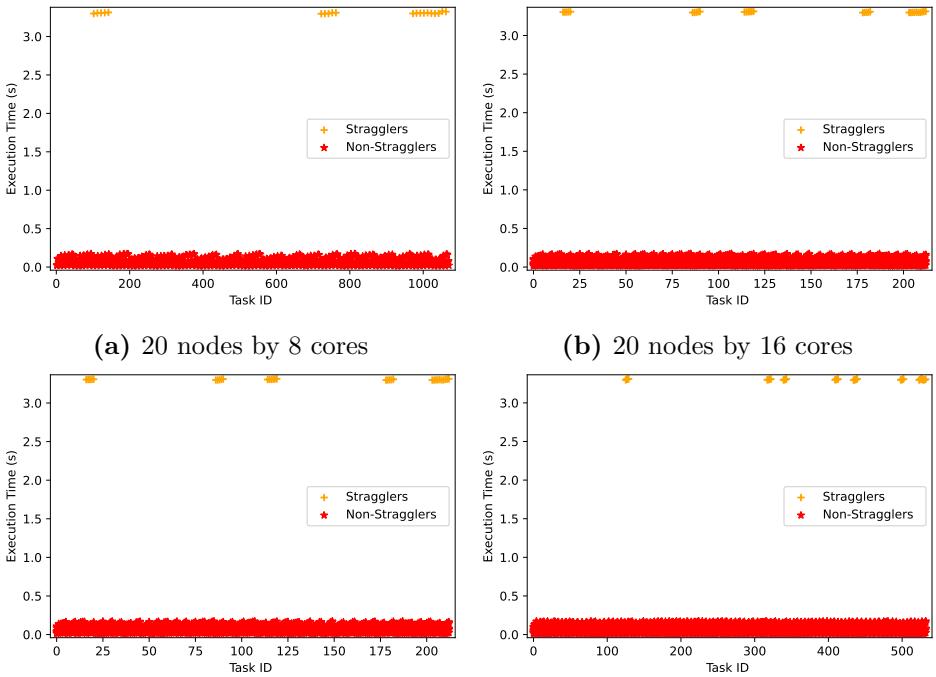
Nevertheless, the Silhouette score (S_x) utilizes an easy-to-evaluate metric to determine the goodness of the clustering. Silhouette score values have a closed range of -1 to 1 [182]. Thus, the silhouette score was chosen for this work.

Algorithm 4 is utilised to identify the suitability of a clustering output for fast and straggler tasks. This algorithm validates the silhouette scores after the clustering exercise. It utilizes the values to decide whether to process backup tasks or not. For instance, figures 4.6a to 4.6d show very close data clusters which are difficult to ascertain the fast tasks or poor-performing ones. However, figures 4.7a to 4.7d show well defined data clusters which will require rescheduling of the straggler tasks.

Algorithm 4 *Kmeans Clustering Validation Algorithm*

Require: : *Set the* S_i *threshold lower* $-$ *bound as* $Z_x = 0.685$
Require: : *Set the* S_i *threshold upper* $-$ *bound as* $Z_y = 0.99$
1: *Initialize the clustering output as an array* $A[k]$
2: **for** $k = 1$ **to** $A.length$ **do**
3: **if** $S_i > Z_x$ & $S_i \leqslant Z_y$ **then**
4: *Reschedule tasks on available nodes*
5: **else**
6: *Run the tasks on current nodes*
7: **end if**
8: **end for**

The results from the silhouette score are utilized to determine the goodness of the K-means clustering. If the silhouette score is higher than the



(a) 20 nodes by 8 cores

(b) 20 nodes by 16 cores

(c) 40 nodes by 8 cores

(d) 100 nodes by 8 cores

Figure 4.7: Kmeans Clustering of Run Times from Experimental Scenarios

threshold lower-bound value Z_x but less than the threshold upper bound value Z_y; backup tasks are required for the data cluster with the straggler tasks as seen in line 3 . Otherwise, no intervention is applied to the task executions. The remainder of the tasks are then processed on their original nodes as seen in line 6.

Our silhouette score threshold lower and upper bound values were determined from several clustering experiments carried out on our data set to ensure that the range given satisfies all possible scenarios.

Let us now discuss the evaluation of the implementation of the job improvement strategy.

4.3 Evaluation of *Haspeck*

The consistent global snapshot strategy was assessed through three major experiments to prove its applicability. They are (i) strategy implementation overheads experiments (ii) job performance experiments (iii) evaluation with Baseline Methods. The experiments enabled us to draw the necessary conclusions on the benefits of using our approach. The experiments aimed to

detect and process straggler tasks as backup tasks during job processing to improve job performance. Snapshots of task run times were captured during job processing which were classified with the K-means clustering algorithm into 2-k categories (fast and straggler tasks). The straggler tasks were then processed as backup tasks on available nodes.

The experimental setup is described in the next sub-section.

4.3.1 Experimental Setup

A laptop (AMD Ryzen (TM)) 7- 4700U with Radeon Graphic, CPU@ 2Mhz, (8 CPU), 16GB, Ubuntu 20.04 LTS was used for the evaluation of our approach. The following objectives were considered in order to achieve the goal of the experiment:

- To determine the commencement time of job processing.
- To determine the completion time of job processing.
- To capture snapshots of task execution times.
- To capture task run times at specific intervals.
- To terminate straggler tasks.
- To restart straggler tasks on available nodes.

The first two bullet points foster the determination of the overheads introduced by our approach. The last four bullet point ensures the measurements of the jobs performance improvements. The termination and restart of the straggler tasks reduces the high consumption of system resources.

The experiment was conducted on our extension of HDMSG MapReduce (a MapReduce simulator with Simgrid as the main backbone) available on GitHub [1]. HDMSG was designed to run on single computer to simulate MR Hadoop cluster behaviours, and as such no server was used in our experiment. The choice of HDEXT was informed by the application of our *MaReClass* framework. *MaReClass* recommended HDEXT as capable of simulating most MR experiments (discussed in sub-section 3.3.1.12).

With respect to core MR simulations, simulators in general, have not earn new features in the past seven or more years, therefore, it was feasible to apply HDMSG for our approach. Although, there are few MR simulators

[1] `https://github.com/EbenezerKomlaGavua/MapReduce_Snapshots`

with extensions to other features such as streaming and IoT; the core behaviours relevant for our research have not changed. This makes our choice for HDMSG suitable for this experiment since it is a simulator capable of simulating MR behaviours such as speculative execution and data replication. Therefore, we extended it with extra methods to make it capable for running our experiments. Also, what was utilised for our experiment was more advanced than the original HDMSG. Furthermore, HDMSG runs on a top of the field simulator (Simgrid with regular updates; the latest being v3.32 (e8d2ff8) updated in Oct 4, 2022).

In order to utilise HDEXT for the development of our *Haspeck* solution, a couple of features had to be added to the simulator to make it applicable. Several methods and classes were created for specific functions. A task monitoring method was created to monitor the tasks running on the nodes on the MR Hadoop cluster. This method was responsible for terminating long running tasks. This method is not a node but a method which utilises the APIs of the Simgrid simulator to monitor the task processing speeds on nodes. A task rescheduling method was created to move the terminated tasks to available nodes for reprocessing as backup tasks. This method took over from the default task scheduling algorithms (FIFO scheduler) whenever there was the need for task reprocessing. A snapshot capturing method was created to capture the start times and completion times of tasks on the nodes. These captured run times were saved on text files and used for the kmeans clustering.

A disruption injection (extra task injection) method was created to send extra (additional) tasks unto arbitrary nodes to serve as background noise. These extra tasks (created randomly) caused actual the map or reduce task to have longer run times on their respective nodes. The intensity of the extra tasks range between twenty-five percent to ninety-five percent. The intensity was increased gradually to correspond with the duration of the mapper execution times. The costs for the extra tasks were determine by the equation 4.2.

$$C = C_F * (hdfs\_chunck\_size\_bytes/mappers) \tag{4.2}$$

Where C is the positive rational numbers for processing the extra cost for a specific mapper execution time. The C_F (Calibration factor) is a positive rational number that determines how long an extra task should run to cause sufficient disruption. The Calibration factor is an integral part of the HDMSG simulator which is employed to create task costs. The $hdfs\_chunck\_size\_bytes$ is the block size and $mappers$ is the positive rational number for the number of mappers per node.

Table 4.2: Set up of HDMSG-EXT

Features	**HDMSG-EXT**
No. of Nodes	20+
No. of Cores	8+
Mappers per node	5+
Reducers per node	5+
No. of Tasks	5+
Block Size(MB)	128+
File Input Size(MB)	1024+
Bandwidth	1Gbps+

The methods for the creation of map and reduce tasks were extended to foster the scalability of the framework. A node scheduling class was created to foster the chronological processing of data nodes to enable the capturing of snapshots.

Ten-equal-length smaller mappers were created (from bigger mappers) as checkpoint barriers to enable the snapshot capturing with simgrid. This was done to ensure that the snapshot algorithm could capture the start-times and completion-times of the smaller mappers. The experiments required that the timelines of task processing were captured. Therefore, the checkpoint barrier allowed the job processing performance of larger mappers to be monitored and captured as snapshots.

In setting up the experiments, the infrastructure of HDEXT with Simgrid were defined. The infrastructure was defined in terms of the following: the number of nodes, CPU cores, bandwidth, latency metrics, and the nodes' speed (in a picluster.xml file). Additionally, the number of mappers and reducers, file input size (in megabytes), and block size (HDFS chunk size in megabytes) were configured (in a config.txt file) to foster MR computations as seen in table 4.2. Table 4.2 shows the upgrades made to the original version of the HDMSG simulator to enable us perform our experiments.

Traffic is generated by the HDMSG.c file at the instance of job processing. The distributedHdfsChunks class of the simulator causes tasks to be created. These tasks are created via the calibrations of the parameters set in the config file, HdmsgHost.c and HdmsgHost.h files of the simulator. The tasks are forwarded to the nodes configured in the picluster file for processing to commenced. The map tasks are processed first, which are followed by shuffling and reduce tasks. The output of task processing is generated when all the tasks created are completely processed.

To determine real life MR cluster infrastructure and application config-

Table 4.3: Survey Table 1

Features	**SandBox Values**	**Standard Values**	**Advance Values [70]**
Hadoop Cluster Mode	Fully Distributed		
Number of nodes	2+	4 - 10+	12+
Network Bandwidth (Gbps)	1	2	10
Total HDFS Capacity	128GB	8 - 80 TB	144TB
Disk Capacity	32 GB	256 GB - 1 TB	1.2TB
Total System Memory(GB)	16	64 GB	128
CPU Speed (Ghz)	2-2.5		2.5 - 3.5
Logical CPU cores	16	24 - 32	48
No. of Mappers	11+	16-21+	32+
No. of Reducers	8	8-19+	22+
Block Size (MB)	128	256	

urations, two surveys about Hadoop cluster requirements were carried out. The survey focused on identifying typical Hadoop configurations and organizations actively utilizing hadoop clusters for their data processing in industry.

Several keywords such as hadoop clusters/ requirements, industry hadoop cluster infrastructure/ setup/ configurations were employed on search engines to locate current MRH cluster configurations.

Hadoop cluster configurations such as basic or standard deployments, advances deployments, Hadoop cluster hardware recommendations for batch processing, in-memory processing, medium data size and large data size were identified from the first survey.

The first survey found that the most used CPU speed was 2-2.5Ghz, data block sizes were between 128-256MB, network bandwidth was 1-10Gbps, cluster nodes was four to forty, number of mappers and reducers were mostly five to twelve per node, disk capacity range was 32 GB to 1.2TB and total system memory was 16-512 GB. All Hadoop cluster configurations modes were fully distributed as seen in tables 4.3 to 4.5.

Table 4.4: Survey 1 Table 2

Features	**Titanwolf Processing Values [155]**	**Hadoopilluminated Values [70]**	
		Med. End Values	**High End Values**
Hadoop Cluster Instal. Mode	Fully Distributed		
No. of Nodes	20	18	12
Network Bandwidth	10 Gbps	1 Gbps	10 Gbps
Disk Capacity	1.2TB	4 TB	36 TB
Total System Memory	64 - 512GB	16 GB	48 GB
CPU Speed	2-2.5Ghz		
Logical or Virtual CPU Cores	8	8	12
No. of Mappers	5		8
No. of Reducers	5		8
Block Size	256MB	128MB	256MB

The second survey found over one hundred and twenty top companies actively utilising Hadoop clusters from several websites. Notable companies amongst the list include Alibaba, AOL, Yahoo, Spotify, Last.fm, Ebay, University of Glasgow-Terrier Team and Criteo. From this list, the modal CPU cores per node was eight and the modal cluster nodes was forty as seen in table 4.6.

The findings of the survey fostered the selection of four infrastructure scenarios (displayed in table 4.7) for our experiments. The experimental scenarios comprise data nodes that ranges from twenty to one hundred nodes. The range of CPU cores was eight to sixteen. Aside the values displayed in the infrastructure scenarios table, network bandwidth of 10Gbps was simulated for all infrastructure scenarios. Thee smallest data block size employed was 128MBs. All the experiments were run on fully distributed hadoop cluster mode to ensure conformance with industry standard.

In terms of modelling real life applications on the above infrastructures, the details below were identified from the survey. The number of mappers

Table 4.5: Survey 1 Table 3

Features	Dzone [50]		Packtpub [120]
	Batching Values	**In-Memory Values**	**Batching Values**
Hadoop cluster instal. mode	Fully Distributed		
Number of cluster nodes	10		20
Network Bandwidth	10Gbps		
Total system memory	60 GB	76GB	28GB
CPU speed	2-2.5Ghz		
Logical or virtual CPU cores	12	16	8
No. of Mappers	8	12	5-6
No. of Reducers	8		4-5
Block size	128 MB		

per node was obtained via equation 4.3, as stipulated in [2]:

$$Y = \frac{2Cores}{3} \tag{4.3}$$

where Y is a positive rational number that represents the number of mappers per node. $CORES$ is a positive integer which represents the CPU cores per node.

The number of reducers per node was obtained via equation 4.4, as stipulated in [3]:

$$R = 0.95 \times N \times T \tag{4.4}$$

Where R and T are positive rational numbers representing the number of reducers per node and the mapred tasktracker reduce tasks maximum value respectively. T is the maximum number of reduce tasks that will be run simultaneously by a task tracker (2 was used, since it is the default maximum value). N is a positive integer representing the number of nodes running on the cluster.

[2]https://data-flair.training/forums/topic/how-one-can-decide-for-a-job-how-many-mapper-reducers-are-required/

[3]https://hadoop.apache.org/docs/r1.2.1/api/org/apache/hadoop/mapred/-JobConf.html

Table 4.6: Survey Two

SN	**Company**	**Cluster nodes**	**Total CPU cores nodes**	**Core per Node**	**Ref.**
1	AOL	150	300	2	[161]
2	Last.fm	100	800	8	
3	Spotify	1650	43000	26	
4	eBay	532	4256	8	
5	Cornell Uni. Web Lab	100	800	8	
6	Alibaba	15	120	8	[142]
7	RightNow Technologies	16	128	8	
8	Socialmedia.com	14	112	8	
9	Criteo	2000	48000	24	
10	Cooliris	15	120	8	
11	Yahoo	4500	36000	8	[48]
12	LinkedIn	4100	82000	20	
13	Rackspace	30	60	2	
14	Facebook	1400	11200	8	
15	Fox Audience Network	140	1120	8	
16	Neptune	200	1600	8	[124]
17	Rapleaf	80	640	8	
18	Uni. of Twente, DB Group	16	32	2	
19	Uni. of Glasgow - Terrier Team	30	120	4	
20	Contextweb	50	400	8	

The File Input Size (in MB) for our experiment was obtained via equation 4.5

$$F = 512 + (128 * (M - 4)) \tag{4.5}$$

Where F and C are positive rational number representing the file size in MB and the total number of mappers required for processing the file input respectively. The five hundred and twelve (in MB) is the original input file size of HDMSG and the one hundred and twenty-eight (in MB) is the block size for the our extension (HDEXT). The default number of mappers on HDMSG architecture is four.

Haspeck applies to both mappers and reducers. However, the evaluation was mostly centred on mappers; since the mappers and reducers showed the same workload pattern and behaviours after a series of initial experiments

Table 4.7: Experimental Set up

Features	Scenario 1	Scenario 2	Scenario 3	Scenario 4
No. of Nodes	20	20	40	100
No. of Cores	8	16	8	8
Mappers per node	5	5	11	5
Total Reducers	38	38	76	190
Total Mappers	107	213	213	533
File Input Size(MB)	13696	27264	27264	68224

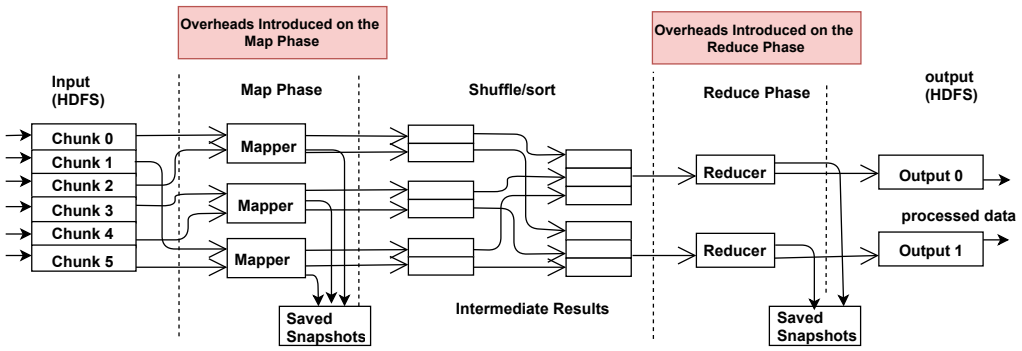


Figure 4.8: Overheads Introduced by Our Approach on MapReduce

of applying *Haspeck*. This is irrespective of the fact that the map function is commutative, and reducers apply a reducing function which is mostly an associative and commutative. As such, the reducers were not ignored in the development of *Haspeck*.

4.3.2 Determining the Overheads of Haspeck

This experiment determined the overheads introduced into the infrastructure by the implementation of our approach. The overheads were caused by the effects of the snapshots capturing process on the infrastructure as seen in figure 4.8. The comprehension of the effects of the overheads fosters the appreciation of the challenges and benefits in applying *Haspeck* on MRH.

The experiment was conducted on the four data centre scenarios discussed in sub-section 4.3. The input file size was applied in modelling the execution time of the mappers and reducers. The calibration for processing duration

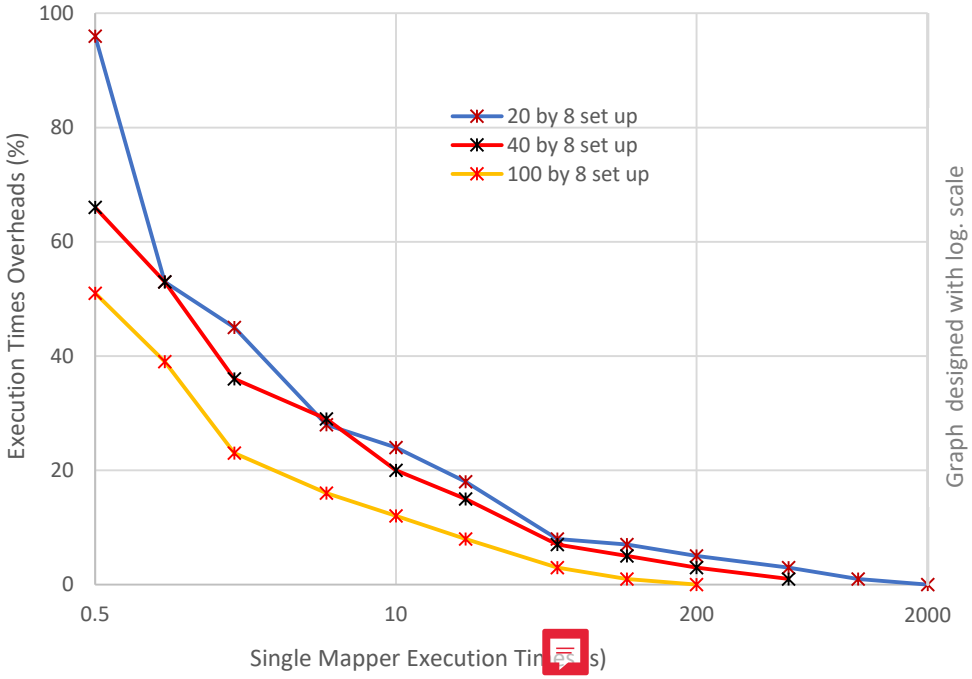


Figure 4.9: Cluster Overheads

of a mapper or a reducer was achieved via the get map cost method (for mappers) and get reduce cost method (for reducers) of the simulator. This method receives inputs from the file size, calibration factor, flops per mb and the speed of a host. These inputs are utilized to set the duration for mappers or reducers. As such, larger files had increased job processing duration while smaller files had a lesser processing periods.

Mapper tasks with execution times from 0.5 to 2000 seconds were utilized. The range for the experiment was derived via the multiplication of the single values of one, two and five with the power series of ten. The value of negative one produced 0.5 seconds and we scaled the executions until the graph converged at 2000 seconds. This process was applied in order to obtain a scalable range of task run times. The experiment was executed until the values converged at 2000 seconds.

Since our approach involves capturing snapshots during the job processing of mappers (i.e. smaller mappers created as checkpoint barriers discussed in 4.3.1)), two measurements were taken. These are (*i*) the commencement of jobs processing and (*ii*) The completion of the processing of jobs. To determine the overhead on a single mapper, the differences between the *completion times* of the processed portion of the task and the *start times* of

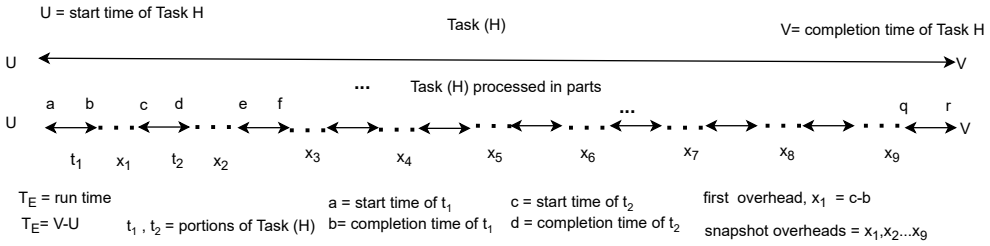


Figure 4.10: Strategy Overheads

the next portion of that same task are determined (that is, the period for snapshot capturing). The summation of the differences of these values (i.e. differences between *completion times* and *start times*) is subtracted from the task's run times (which is the ten-equal-length smaller mappers) as shown in figure 4.10. The value realized is the overhead on a single mapper as shown in equation 4.6 and figure 4.9.

$$Q^H = 100 - \frac{100(T_E^H - (\sum_{i=1}^{n-1} t_c^{H,i} - t_s^{H,i}))}{T_E^H} \quad (4.6)$$

Where Q^H is the overhead of applying our approach on a mapper H in percentage. T_E^H is the task runtime of the given mapper. $t_s^{H,i}$ is the time the i^{th} snapshot of the mapper H was started to be captured. Similarly, $t_c^{H,i}$ is the time when we finished capturing the snapshot of the same task. Equation 4.6 is exemplified in figure 4.10.T_E^H is obtained from subtracting U from V. The letters a and c are the start times of the first two smaller mappers, whilst b and d are the completion times of the first two smaller mappers. Hence subtracting b from c produces the first gap (x_1) introduced because of our approach. These gaps x_1 to x_9 are summed up and divided by the task run times to generate the overhead of a task.

The overheads of a single mapper were measured on the four data centre scenarios as seen in tables 4.8 and illustrated in figure 4.9.

Scenario 20N×8c The impact of applying *Haspeck* was gradual. The overheads were high at the initial stages of the experiment. However, the impact of applying *Haspeck* caused the high overheads to reduce gradually with longer runtimes, as seen in figure 4.9. Therefore, in such small infrastructures, our approach is only advisable to use with long run times.

Scenarios 20N×16c and 40N×8c exhibited similar overhead behaviours during task runs. Therefore, only the 40 by 8 set up was represented in

Table 4.8: Strategy Implementation Overheads

	Scenario overheads (%)			
Map time (ms)	20N×8c	20N×16c	40N×8c	100N×8c
0.5	96	66	66	51
1	53	53	53	39
2	45	36	36	23
5	28	29	29	16
10	24	20	20	12
20	18	15	15	8
50	8	7	7	3
100	7	5	5	1
200	5	3	3	∼ 0
500	3	1	1	∼ 0
1000	1	∼ 0	∼ 0	∼ 0
2000	∼ 0	∼ 0	∼ 0	∼ 0

figure 4.9. The initial overheads observed were 1.5% lower (relatively) than the 20N×8c data centre scenario. The figure shows that these larger infrastructure scenarios converged faster than the previous. This figure shows that the overheads of larger data centres improve better than smaller ones when *Haspeck* is applied. Also, the overheads of applying our approach with long run times have a higher chance of improving than smaller ones.

Scenario 100N×8c demonstrates how *Haspeck* deals with larger data-centres. The overhead further reduced over the above scenarios. The initial overheads were 1.9% lower (relatively) than scenario 20N×8c and 1.3% lower (relatively) than the other two scenarios. Also, as the task run times increased, the overheads reduced drastically. Therefore, applying *Haspeck* to this scenario shows that initial overheads are mostly lower in large data centres. Additionally, it shows that with large configurations, the overheads reduce faster with long mapper run times than in the other scenarios.

The costs on the four algorithms of *Haspeck* appeared fixed and proportional at the onset of data processing. However, as more jobs were processed, the costs became variable, causing the overheads to reduce with time as seen in figure 4.10. Moreover, the graphs was drawn relative to the disruption introduced. Therefore, as more jobs are being processed, the disruption reduced which caused the overheads to reduce.

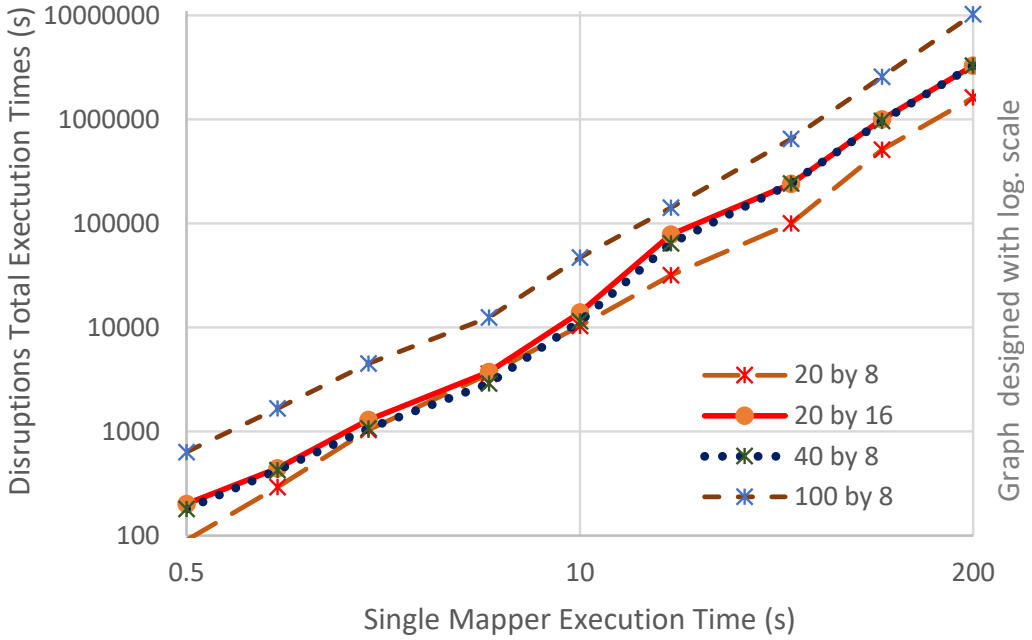


Figure 4.11: *Disruptions* Introduced on Experimental Scenarios

Therefore, from our experiments and industry surveys, we recommend that infrastructures with fourteen to twenty cluster nodes and eight cores should use our scenario 20N×8c data centre configuration. Infrastructures with twenty-five to thirty five cluster nodes with either eight or sixteen cores should use our 20N×16c data centre configuration. Infrastructures with forty to sixty cluster nodes should use our 40N×16c data centre configuration. Finally, infrastructures with one hundred to one hundred and fifty cluster nodes should use our 100N×8c data centre configuration. Moreover, our *Haspeck* solution is applicable to all the above configurations and the above recommendations can be customized to suit user preferences.

4.3.3 Job Performance Experiments

This experiment determined the impact of *Haspeck* on job performance. Four measurements were taken to evaluate our approach. These are:

- Total execution times when there was *no disruption* on the MapReduce set-up (i.e., a dedicated hadoop cluster scenario).
- Total execution times when *disruptions* were introduced on arbitrarily nodes on the infrastructure (as discussed in sub-section 4.3.1). These disruptions were created to interfere with job processing, so that the task will have long run times than expected (this experiment was meant to represent a hadoop cluster hosted in a multi-tenant environment). These *disruptions* were introduced via the running of extra tasks on arbitrarily nodes which were not linked to the original map or reduce tasks. The extra tasks were designed to consume extra system resources during the map and reduce phase. The *disruptions* represent background services, IO contentions or uneven distribution of resources on data nodes for industry research.
- Total execution times when tasks were terminated and processed as backup tasks *resuming* from their snapshots on a different host (this represents situations when mappers can restore their mid-execution states (*reschedule*)). This is applied by applications with the capabilities of storing their states during data processing. When such applications get terminated abruptly due to factors contributing to speculative execution; the applications resume on available nodes and continue from the point their processing was halted.
- Total execution times when tasks were terminated and processed as backup tasks (*restart*) on a different host (providing insight into applications which cannot take advantage of state restoration)

These measurements were utilized to draw the graphs shown in figure 4.12. The graphs of *no disruption*, *reschedule* and *restart* were drawn relative to the *disruption* graphs which are shown in figure 4.11. Because we wanted to observe the job performance improvements when *Haspeck* was applied in spite of the disruptions introduced into the system. As such, the graphs are not a time-series plots.

The *disruption* is at the 100% mark on figure 4.12; and its effect is completely reduced at the horizontal line at 0% on each graph. Therefore, job performance improvement of the graph is seen by the reduction of the heights

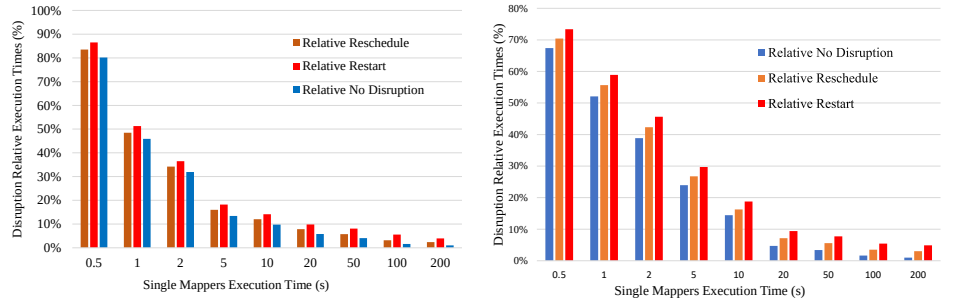
of the bars in the figure towards the 0% mark. The details of the impact of *Haspeck* on the various scenarios are discussed below.

First, figure 4.12a displays the impact of *Haspeck* on the 20N×8c data centre scenario. The job improvement on this data centre was gradual as seen in the figure. In relation to the *disruption* graph, the slope began from above 80% at 0.5 seconds and improved to below 40% at 5 seconds. The job performance improved to below 15% at 10 seconds. The application of *Haspeck* caused job performance to improve from 10% to 2% from 20 seconds to 200 seconds relative to *disruptions*. Furthermore, figure 4.12a showed that *reschedule* backup tasks improve better with *Haspeck* than *restart* backup tasks relative to disruption.

Second, scenario 20N×16c data centre demonstrated considerably job performance improvement than the previous data centre; since most of the graphs were below the 80% mark as seen in figure 4.12b. In relation to the *disruption* graph, the slope began from above 73% at 0.5 seconds to below 40% at 5 seconds. The job improvement increased to below 20% relative to *disruption* at 20 seconds. Also, tasks that transfer their states perform better with *restart* than those that cannot. Tasks with long run times exhibited big improvements, as their values were below 10% relative to *disruption* graphs. This means that as the jobs are processed for long tun times, the effects of the disruptions were reduced as the graphs approached the 0% mark. For industry practitioners, it is advisable to apply *restart* for long run times.

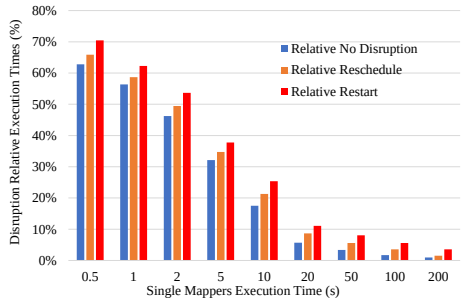
Third, scenario 40N×8c data centre improved more, compared to the previous two scenarios as seen in figure 4.12c. In relation to the *disruption* graph, the job improvement began from above 70% at 0.5 seconds to below 40% at 5 seconds. Job improvement continued to below 5% at 200 seconds.

Fourth, scenario 100N×8c improved much more than all the previous three data centre scenarios. The graph showed a better job performance improvement from below the 60% at 0.5 seconds relative to the *disruption* graph, as seen in figure 4.12d. At 5 seconds, the graph was below 40%. The job improvement continued to 10% and below at 20 seconds. The job performance continue to an average of 2% at 200 seconds. The figure showed that jobs with long run times has higher chances of improvement in this data centre. As most of the graphs were below 10% relative to the *disruption* graph. Also, this shows that they were closer to 0% mark; hence the effects of the disruption introduced were reduced. Also, *reschedule* backup tasks improved much better than the (*restart*) backup tasks. Since the *reschedule* backup tasks have the capability to save their states, as such, they continued data processing when they were moved to other nodes. In contrast, the *restart* backup tasks had to begin all over, which delayed their task processing duration when moved to other nodes.

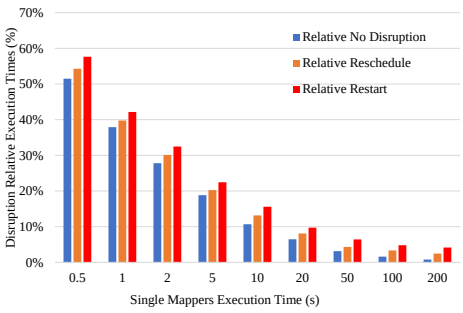


(a) 20 nodes by 8 cores

(b) 20 nodes by 16 cores



(c) 40 nodes by 8 cores



(d) 100 nodes by 8 cores

Figure 4.12: Jobs Improvement Experimental Scenarios (Drawn Relative to *Disruptions*)

In general, the graphs in figure 4.12 show an improvement of *Haspeck* against *disruptions* from 0.5 seconds to 200 seconds, as the height of the bar reduced as the execution times increased. Also, all the four data centres showed an average of 8% job improvement at the 20 seconds mark. This means after the first 20 seconds, jobs on all the data centres perform at an optimum rate with a higher chance of job improvement when *Haspeck* is applied. Let us now discuss the evaluation of our approach with baseline methods.

4.3.4 Evaluation with Baseline Methods

In order to evaluate the performance improvements of *Haspeck* on MRH, selected baseline methods discussed in section 2.6 were executed on our experimental set-up and the results compared. The methods utilised were the Hadoop Naive, Longest Approximate Time To End (LATE) [179] and the Self-Adaptive MR Scheduling Algorithms (SAMR) [33]. Appropriate precautions were taken to ensure that the values produced for analysis were

accurate. These include:

- The application of the exact total execution times for *no disruption* and *disruption* as discussed in section 4.3.3.
- Utilisation of the same data centre configurations applied to *Haspeck* to foster effective comparison.
- The utilisation of the captured snapshots to determine the progress score. Regarding Hadoop Naive, a task is declared as straggler task if its progress score is smaller than the average progress score of all running tasks minus 20%. In the case of LATE, straggler tasks which were determined via the progress score applied in Hadoop Naive, was applied together with a static variable (speculative cap which is analogous to 10% in its algorithm). These straggler tasks were determined on nodes with progress rate greater than the *slowNodeThreshold* as stipulated in [179]. In the case of SAMR, weights were computed and stored in an xml file after job processing. These weights were utilised together with the slowNodeThreshold applied by LATE to determine the straggle tasks as stipulated in [33].
- Jobs were processed with the base line methods for the same duration as our approach to ensure efficient comparison.

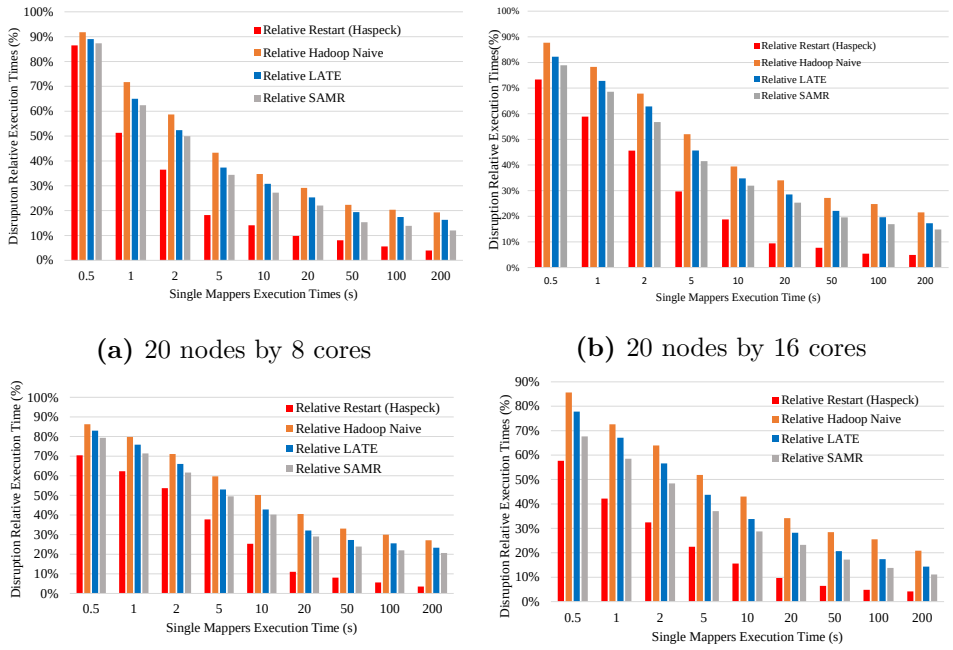
The measurements recorded after these experiments were utilised to draw the graphs in figure 4.13. Let us now discuss the observations made after the experiments.

4.3.4.1 Experimental Discussion

Experiments were carried out on the four data centre configurations discussed in sub-section 4.3.3.The measurements were compared with *Haspeck*. The graphs of all the methods were drawn relative to the *disruption* graphs as seen in figure 4.13. Therefore, job improvements is seen by how close an approach is to the 0% mark.

In determining the job improvements via the application of *Haspeck* to the selected Baseline methods; we first computed the averages of all the relative job improvements. Then we computed the ratio of the average relative job improvements of a particular baseline method to the ratio of the average of the relative job improvements of *restart* (i.e. *Haspeck*) as seen in equation 4.7.

$$JIR = (ARJIB/ARJIH) \tag{4.7}$$



(a) 20 nodes by 8 cores

(b) 20 nodes by 16 cores

(c) 40 nodes by 8 cores

(d) 100 nodes by 8 cores

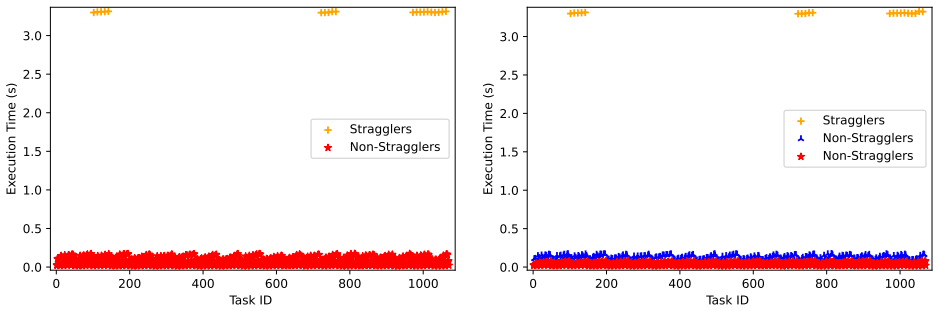
Figure 4.13: Comparison with Baseline Methods (Drawn Relative to *Disruptions*)

Where JIR is a positive rational number representing the Job Improvement Ratios. ARJIB is a positive rational number representing the Average Relative Job Improvement of Baseline Methods. ARJIH is a positive rational number representing the Average Relative Job Improvement of *Haspeck*.

Let us now highlight the job improvements observed with the data centre configurations.

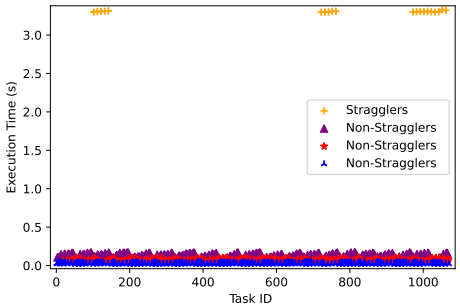
First, on the 20N×8c data centre configuration, *Haspeck* showed 1.67, 1.51 and 1.39 times average job improvements over Hadoop Naive, LATE and SAMR as seen in figure 4.13a. Second, on the 20N×16c data centre configuration, *Haspeck* showed 1.70, 1.52 and 1.40 times average job improvements over Hadoop Naive, LATE and SAMR as seen in figure 4.13b. Third, 1.72. 1.54 and 1.43 times average job improvements were observed over Hadoop Naive, LATE and SAMR in the 40N×8c data centre configuration. This is seen in figure 4.13c. Fourth, the 100N×8c data centre configuration, *Haspeck* showed 2.18, 1.84 and 1.56 times average job improvements over Hadoop Naive, LATE and SAMR as seen in figure 4.13d.

In conclusion, larger data centres have a higher chance of improvement when applying our approach. *Haspeck* works better with larger data centres

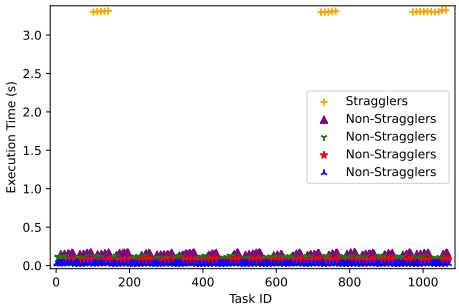


(a) 2-k Data Clusters

(b) 3-k Data Clusters



(c) 4-k Data Clusters



(d) 5-k Data Clusters

Figure 4.14: Kmeans Data Clusters of Tasks Execution times from 20 Nodes by 8 Cores Data Centre with Disruption

because their sizes fosters scalability with long run times, which also ensures reduction in system overheads.

4.3.5 Disruption identification with Kmeans Clustering

The task run times captured during the experiments were utilized for the K-means clustering. Two categories of results were observed after the clustering. Disruption-induced and disruption-free categories. The straggler tasks formed the disruption-induced data clusters are seen in figures 4.7a to 4.7d. The large magnitudes of the straggler tasks, enabled k-means to properly create the two categories.

To determine the number of k-clusters suitable for our work, we generated several clusters from our experimental data set. A selected twenty data clusters, generated from disruption-free and Disruption-induced categories is shown as seen in figures 4.15 and 4.16. A visualization of some of the data clusters (k=2 to k=5) for both categories is seen in figures 4.6 and 4.14.

Table 4.9: KMeans Clustering Silhouette Scores of Disrupted Data Clusters

Figure Number	Silhouette Scores
4.14a	0.985
4.14b	0.688
4.14c	0.604
4.14d	0.641

The silhouette scores of figures 4.6a to 4.6d show a reduction in value as the number of data clusters increased as seen in table 4.1. These values are closer to our silhouette score threshold lower-bound value. Furthermore, the values of the silhouette scores of figures 4.14a to 4.14d showed similar reduction as the number of data clusters increased as seen in table 4.9.

Although the visualization shows stragglers situated at the top of the sub-figures in figure 4.14. The silhouette scores is confusing. This is due to the fact that the euclidean distance between the centroids decreases as the number of data clusters increases. This affects the decision making of the silhouette scores and when to process straggler tasks are backup tasks (generally, affecting the efficiency of *Haspeck*).

In general, the time bound for k=2 to k=20 data clusters are the same, as such, the choice of k=2 for *Haspeck* inures to efficient decision making.

4.4 Summary

This chapter focused on designing and implementing a job performance solution for MapReduce Hadoop. First, we reviewed a few concepts related to MapReduce and Hadoop to provide the necessary background for our *Haspeck* solution. Second, the three algorithms that form the intricate components of our solution were described in detail. They are snapshot capturing, task and node performance monitoring and task instance monitoring algorithms. K-means clustering algorithm was applied to classify the captured snapshots into straggler and non-straggler tasks. Silhouette score was implemented in the K-means to validate the data clusters and also to determine when to process straggler tasks as *backup tasks*. Our solution was evaluated after an industry-specific survey on Hadoop implementations. This was to ensure that the solution was applicable to industry and real life. Several experiments were conducted, including the strategy implementation, job performance experiments and evaluation with baseline methods. The experiments revealed that implementing *Haspeck* reduced overhead as the mapper run times increased. Also, the job performance improved on mappers with long run times.

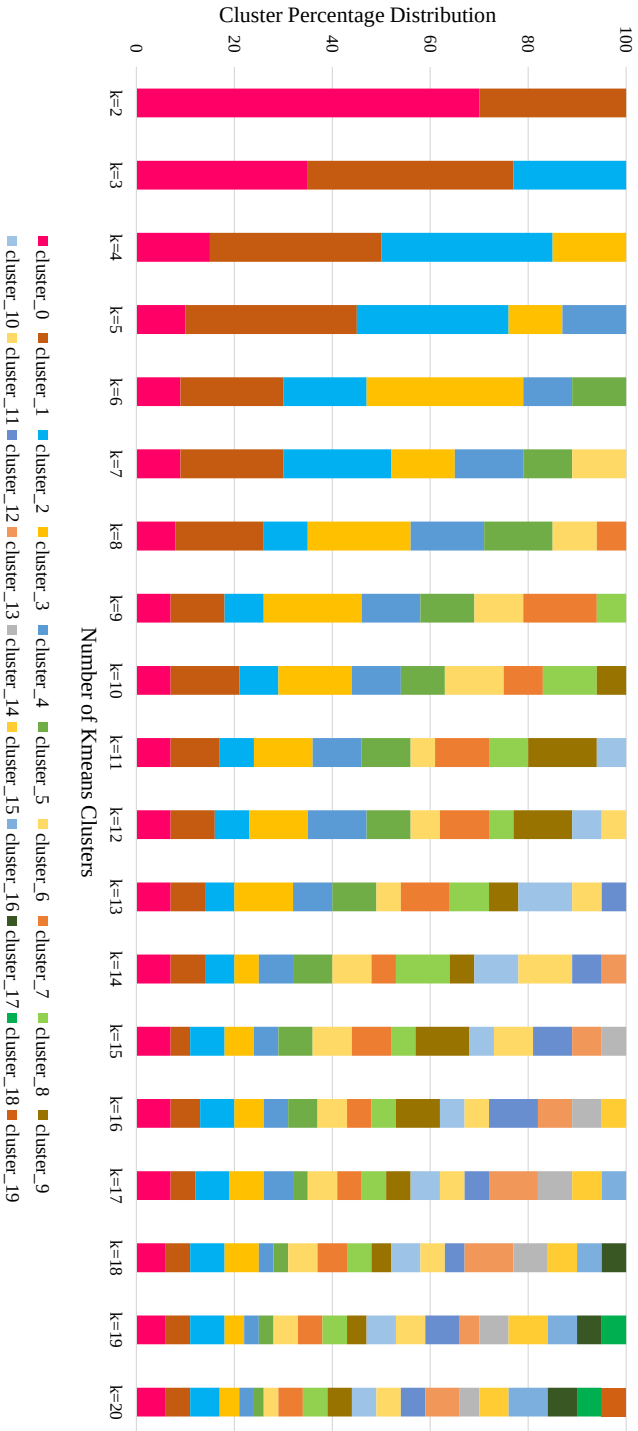


Figure 4.15: Kmeans Clusters on a No-Disrupted Data Center Scenario

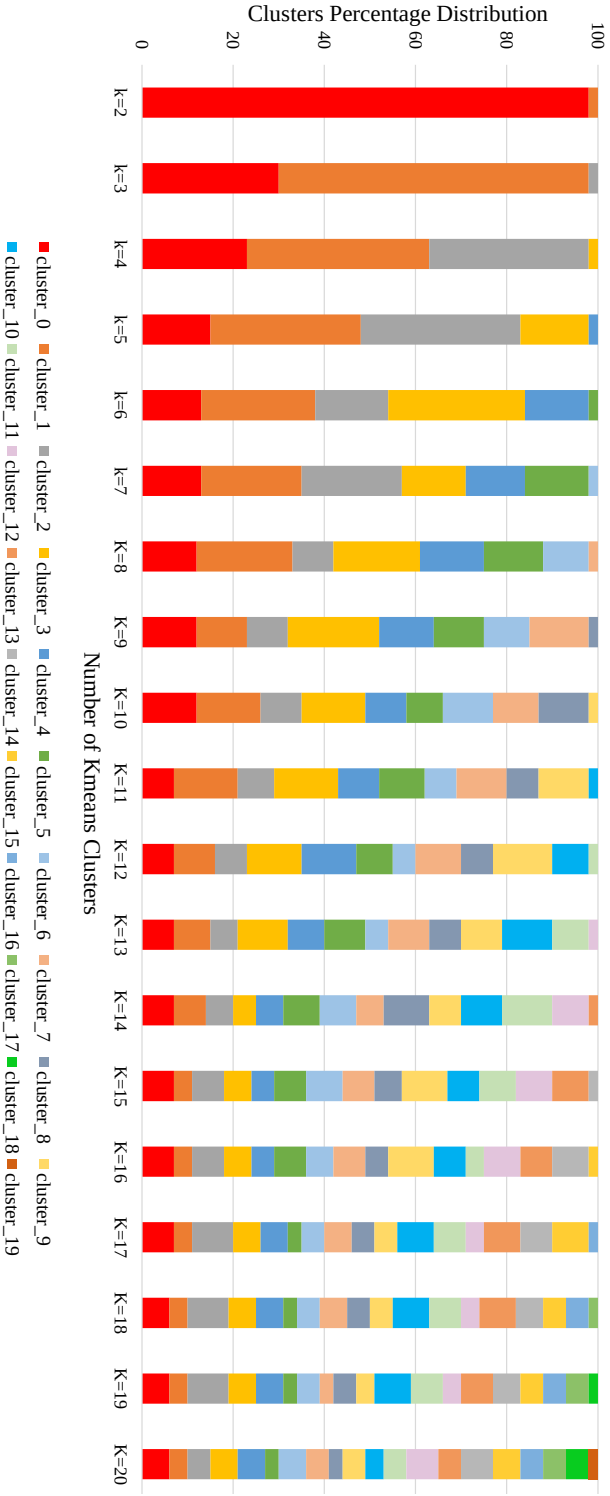


Figure 4.16: Kmeans Clusters on a Disrupted Data Center Scenario

Furthermore, our evaluations helped us to conclude that, larger data centres with longer run times have a higher chance of improvement in applying *Haspeck*. These larger data centres operate by scaling available resources to meet user demands which is implemented in auto-scaling mechanisms on clouds. As discussed in previous chapters 2.7, auto-scaling approaches are designed and implemented on MRH and other cloud infrastructures to foster resource provisioning to meet user demands. This synchronises with *Haspeck* aim for the improvement of job performance.

In the next chapter, we shall design an ASM Model to analyse the behaviours of cloud auto-scaling mechanisms. This is achieved via investigating auto-scalers developed on DISSECT-CF.

Chapter 5
Modelling Auto-Scaling Mechanisms in Clouds

5.1 Introduction

In the previous chapter, we have discussed several job improvement approaches on MR. In those discussions, we saw that, the primary objective of these approaches is to enhance resource provision to meet user demands whilst reducing costs on the clouds. Furthermore, we discussed the benefits of applying these approaches on cloud frameworks including MR.

In this chapter, we focus on analysing the behaviours of auto-scaling mechanisms on cloud mechanisms. Cloud mechanisms are the essential components that make up the architecture of a cloud environment. These mechanisms include several techniques and technologies to ensure the availability, flexibility and cost savings of the resources provided over a distributed network.

Our discussion in section 2.8 showed that several ground models for clouds and distributed systems exits, however, these models lack components for extension to auto-scalers. The models did not discuss the job execution phases of auto-scalers; neither did they highlight the specific features for vertical and horizontal scaling of resources. Also, the existing ground model did not highlight state changes during job processing; neither did they evaluate specific features of auto-scalers (such as VM provision thresholds). The absence of these primary components makes the utilization of these ground models unsuitable for auto-scalers. Due to these gaps, we set out to develop a ground model and its refinements for evaluating auto-scalers.

The development of our *Astam* model first focused on investigating the behaviours of auto-scalers on DISSECT-CF and literature. The investigations serve as an inspiration towards the identification; and the general com-

prehension of the behaviours of shared components required for our model. As such the best way to achieve this was to see several auto-scalers. These components are documented and utilised for the design of *Astam*.

Once the ground model was established, we validated and refined it to ascertain its suitability and applicability to other auto-scalers. The modeling was done to show that, although the auto-scales were developed on different cloud frameworks, they exhibited similar behaviours during job processing.

5.2 Investigating Auto-Scaling Mechanisms

In this section, the auto-scaling mechanisms offered alongside DISSECT-CF are reviewed.

Many auto-scalers are evaluated through simulations. So, our investigations were made of one such simulation environment. DISSECT-CF was chosen for this work because it has been shown through research, to be more efficient for auto-scaling experiments as compared to other simulators. DISSECT-CF (discussed in sub-section 2.4) has five major subsystems which allows the monitoring of auto-scaler behaviours.

In this research, the simulator's infrastructure management system (i.e., which models cloud behaviour with its physical & virtual machines, networking and storage) is the main focus. In building our investigations, the simulator's auto-scaling related examples[1] have been examined. The examination process involved the analysis of the internal components of the simulator designed to ensure the auto-scaling resources. The specific actions taken include:

- Running several experiments with the existing auto-scalers with jobs from the Parallel Workloads Archive [2].
- Observing the the source codes closely before and also at run time to comprehend how the algorithms are put together.
- Extracting the auto-scaling part from the rest.
- Based on the observation, two extra auto-scalers (one multimode and one simple) were created.
- These are presented here as the model.

[1] available at `https://github.com/kecskemeti/dissect-cf-examples` and at `https://github.com/kecskemeti/dcf-exercises`

[2] `https://www.cse.huji.ac.il/labs/parallel/workload/logs.html`.

The auto-scaling mechanisms were built on several components presented in figure 5.1. They can be grouped into two categories (i.e., *simple* and *multimode* auto-scalers). The *simple* auto-scalers respond to demands by increasing or decreasing the VM instance counts according to workload demands. The *multimode* auto-scalers, in addition to exhibiting *simple* auto-scaler features, monitor the VM counts during scaling operations while controlling the utilisation of VMs. VM counts are monitored to assess the general performance of the infrastructure to know when to increase them to meet workload demands or decrease them to reduce consumption of system resources. Also, VM utilisation are controlled via the engagement of a timer and an energy meter which assess the resource usage within specific durations.

All auto-scalers are founded on a handful of classes like the Virtual Infrastructure (VI), JobArrivalHandler, BasicJobScheduler (BJS), JobLauncher, JobToVmSchedulers, GenericTraceProducer (GTP) etc. VI has the dedicated role of managing virtual machines belonging to particular applications. JobArrivalHandlers abstract the application model with the help of replaying customisable parts of pre-recorded workload traces. BasicJobScheduler combines with the JobArrivalHandler's functionality to process jobs. The BasicJobScheduler takes basic job scheduling decisions as it does not have information on other applications compared to the VI which has other classes to supply the necessary information for decision making. Therefore, the BasicJobScheduler uses cluster utilisation patterns to monitor the VM utilisation on the infrastructure. Now let's focus our attention on the available auto-scalers offered with the simulator:

ThresholdBasedVI Mechanism (Threshold) is governed by a lower and an upper threshold. As a *multimode* auto-scaler, it observes virtual machines utilisation and makes decisions based on how it relates to the two thresholds in the VI. It removes VMs that are not used even to the extent of the lower threshold. In contrast, it adds new VMs when the number of VMs in the managed infrastructure are utilised more than the upper threshold.

VMCreationPriorityBasedVI Mechanism (Vmcreate) is a variation of the above approach, but instead of removing VMs, it tries to create them first (i.e., it anticipates growth).

PoolingVI Mechanism (Pooling) is designed to keep a given number of completely unused VMs for newly arriving jobs. As a *multimode* auto-scaler, it extends the VI to accept new jobs anytime during the application run time.

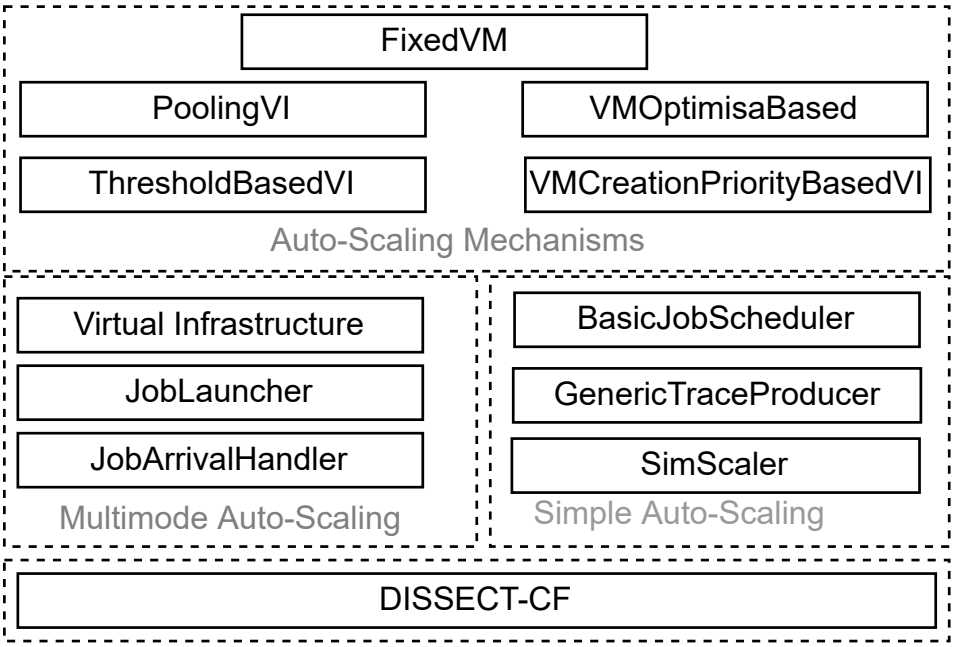


Figure 5.1: Architectural view of Auto-Scaling Mechanisms on DISSECT-CF

VMOptimisationBased Mechanism (Vmopt) allows VMs created for one kind of executable to be repurposed to execute others, fostering VM reuse. Vmopt utilises a reservedset to monitor the VI for any changes. As the number of VM request increases, the available VMs in the reservedset are provisioned. It does not create new VMs.

FixedVM Mechanism (Fixed) is designed to accept jobs for processing and scaling with reduced system resources. It is the only auto-scaler in the simple category. It utilizes a basic scaling mechanism (called *simscaler*) on a production cloud infrastructure. The *simscaler* combines with the GenericTraceProducer (a class responsible for jobs generation) to provision VMs and jobs for job processing. It generates VMs into a simple IaaS infrastructure to process the generated jobs.

The knowledge of these auto-scalers helped us to design our *Astam* model, supported by the ASM theory (discussed in 2.10.1). Now, Let us discuss the design of our *Astam* model.

5.3 Design of Astam Model

The modelling process incorporates all the various stages auto-scalers undergo to provision resources. Our *Astam* model was developed following the ASM refinement method (as design rationale) (discussed in 2.10.1.2), which is presented in 5 steps. They are:

Step 1: Design and Analysis of *Astam's* framework as displayed in figures 5.2 and 5.5. The framework shows our model's ground model for the two categories of auto-scalers. Figure 5.2 shows the basic elements (*universes* interacting with *signatures* with arrows) utilized in designing our model. The arrows represent the relations between the *signatures* and the *universes* while provisioning resources during the multimode or simple auto-scaling. The *signatures* are declared through functions created in accordance with sub-section 2.10.1.3. Figure 5.5 shows *universes* interacting with unidirectional and bidirectional arrows. The bidirectional arrows represents information flow between universes while the unidirectional arrow represent the expected state changes during ASM runs.

Step 2: Design and implement the model's *ASM Transition Rules* to reflect the job execution phases. Five ASM rules were defined and discussed in conjunction with the model refinement 2.10.1.2 and definitions 2 and 4. This is to ensure the provision of details and clarifications of our model as we progress towards lower levels of abstractions.

Step 3: Refine algorithms from the two categories of auto-scalers offered with DISSECT-CF, with *Transition Rules*. This step was modelled simultaneously with the previous step to ensure model coherence.

Step 4: Evaluation of the model (with existing auto-scalers) with the *Transition Rules* and evaluation goals. This process was achieved in conjunction with the ASM refinement method (an ASM benchmark). The evaluation goals were employed to check whether the application of our model to existing auto-scalers produced ASM refinements which are equivalent to our ground model. The evaluation goals were selected in according to the application our transition rules and their alignment to control state ASMs.

Step 5: Model Validation with validation goals on test cases created from existing auto-scalers. The test cases are sets of codes abstracted from the formalized algorithms of our auto-scalers to determine if the algorithms satisfy our ground model requirements (*universes* and *signatures*

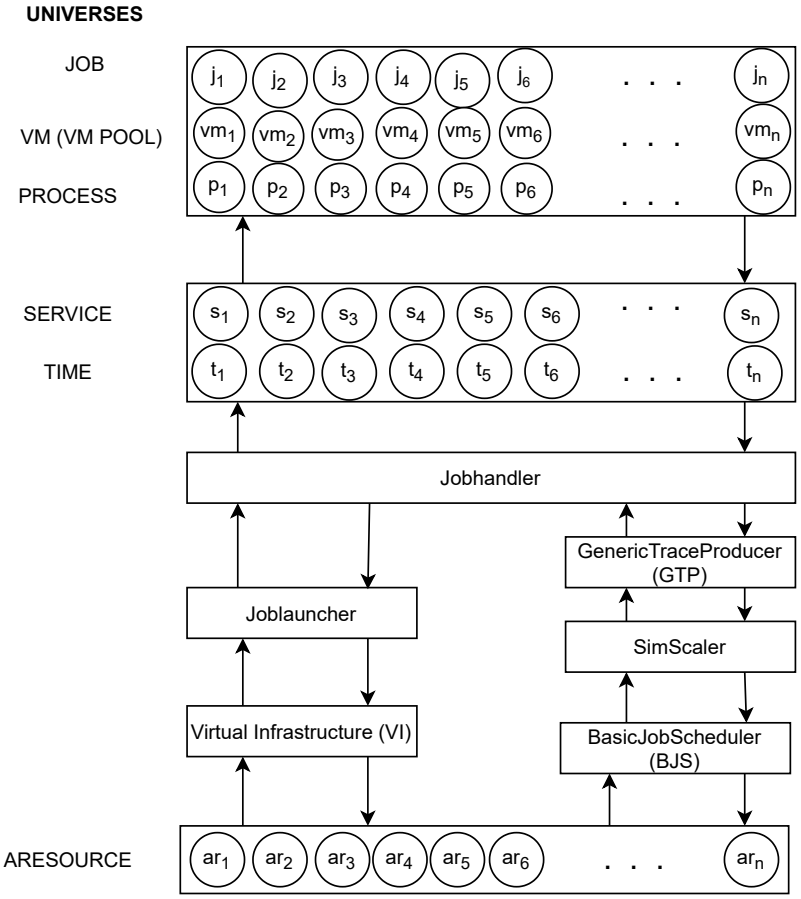


Figure 5.2: Basic elements of the ASM model for Auto-Scaling

correctly. Computational Tree Logic (CTL) formulae were applied for our model's verification. CTL formulae were applied because they are generally utilized for the model checking and verification of ASM models. Also, they provide the logical tools for checking the equivalence of transitions systems.

5.3.1 Astam's Universes

As part of step 1 of the design of *Astam*, let us discuss the universes utilised. These universes are required to develop our ground model, as discussed in sub-section 2.10.1.3. These existing universes (*JOB, ARESOURCE, PROCESS*) required redefinition to make them relevant to *Astam*.

The *JOBHANDLER*: is the universe that processes traces and sends its jobs to a job launcher in the multimode auto-scaling mechanisms. In

the simple auto-scaling mechanism, it is responsible for sending jobs to VMs for processing.

The *JOBLAUNCHER*: is the universe that emits jobs for the multimode mechanisms. This deals with the ordering and timing of new jobs before they are released for processing.

The *JOB*: is the data submitted by a Jobhandler to be executed on a node. It contains binaries of an application, libraries and resource descriptions.

The *ARESOURCE*: Abstract resources represent the major resources required for job processing. These include virtual appliances & infrastructures, cores, cloud service, CPU architecture, memory, and disk requirements. Different categories of resources (i.e. heterogeneous nodes) are also part of *Aresources*.

The VM: is the virtual machine required for processing jobs in an auto-scaled infrastructure.

The *TIME*: is the duration a submitted job must spend being processed by virtual machines. This is usually measured in seconds.

The *PROCESS*: is responsible for ensuring that installed tasks receive the necessary attention from the *Aresources* and the VM.

The *SERVICE*: is responsible for service provision in multimode environments. Services are supplied per user requests at the time on service clouds.

The *VI*: is responsible for managing VMs for applications in multimode environments.

The *SIMSCALER*: is responsible for ensuring basic scaling activities in simple auto-scaled environments.

The *BASICJOBSCHEDULER (BJS)*: utilises clustering patterns to monitor VMs in simple auto-scaled environments.

The *GENERICTRACEPRODUCER (GTP)*: provisions sets of jobs for specified durations in simple auto-scaled environments.

In order for our *universes* to be relatable, there is the need for ASM functions. Let us now discuss the design process of the functions applied in *Astam*. These functions are also part of step 1 of our model's design.

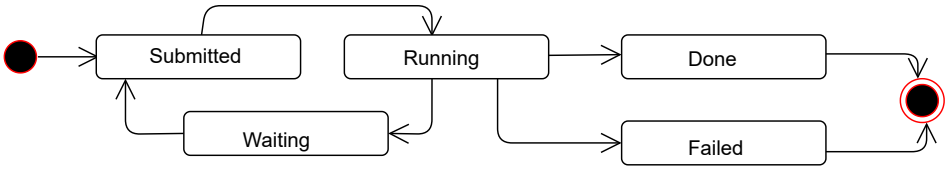


Figure 5.3: Job State Transitions

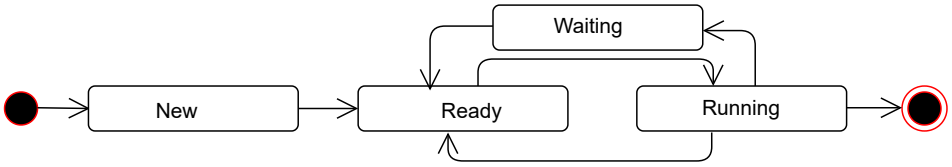


Figure 5.4: Process State Transitions

5.3.2 Astam's Functions

Astam's Functions were created per the specifications of control state ASM (described in definition 2) and supported with *State Transition Diagrams* in figures 5.3 and 5.4. Figure 5.3 shows the job states transitions when mapped to a VM for processing. A job initially mapped to a VM is in the *submitted* state. It is then transformed into a task for processing. When the task receives the attention of the CPU, its state changes to *running*. If the time allocated is sufficient, the job will be fully processed and moved to *done* state. However, if the time runs out, the job is rescheduled for a later time, which moves it to *waiting* until a VM is available for processing. However, if an unexpected event such as system interrupt occurs, a job in the *running* state is moved to the *failed* state.

Additionally, figure 5.4 shows process state transitions. When a process is created to assist in job processing, it arrives in the *new* state. It then transitions to the *ready* state when it has sufficient *aresouces* to commence job processing. The state changes to *running* when a job is installed a task and has began processing it. This state will continue until the task is completely processed. However, if the allocated *aresouces* are exhausted, the task is rescheduled and the process state transitions to *waiting*.

The activities of the functions and the state diagrams illustrate the control (*operation*) refinement steps of an ASM model. Our functions are shown in table 5.1 and discussed below:

JobState depicts the state transitions of job during data processing. Jobs' states transition from submitted to either done or failed as shown in

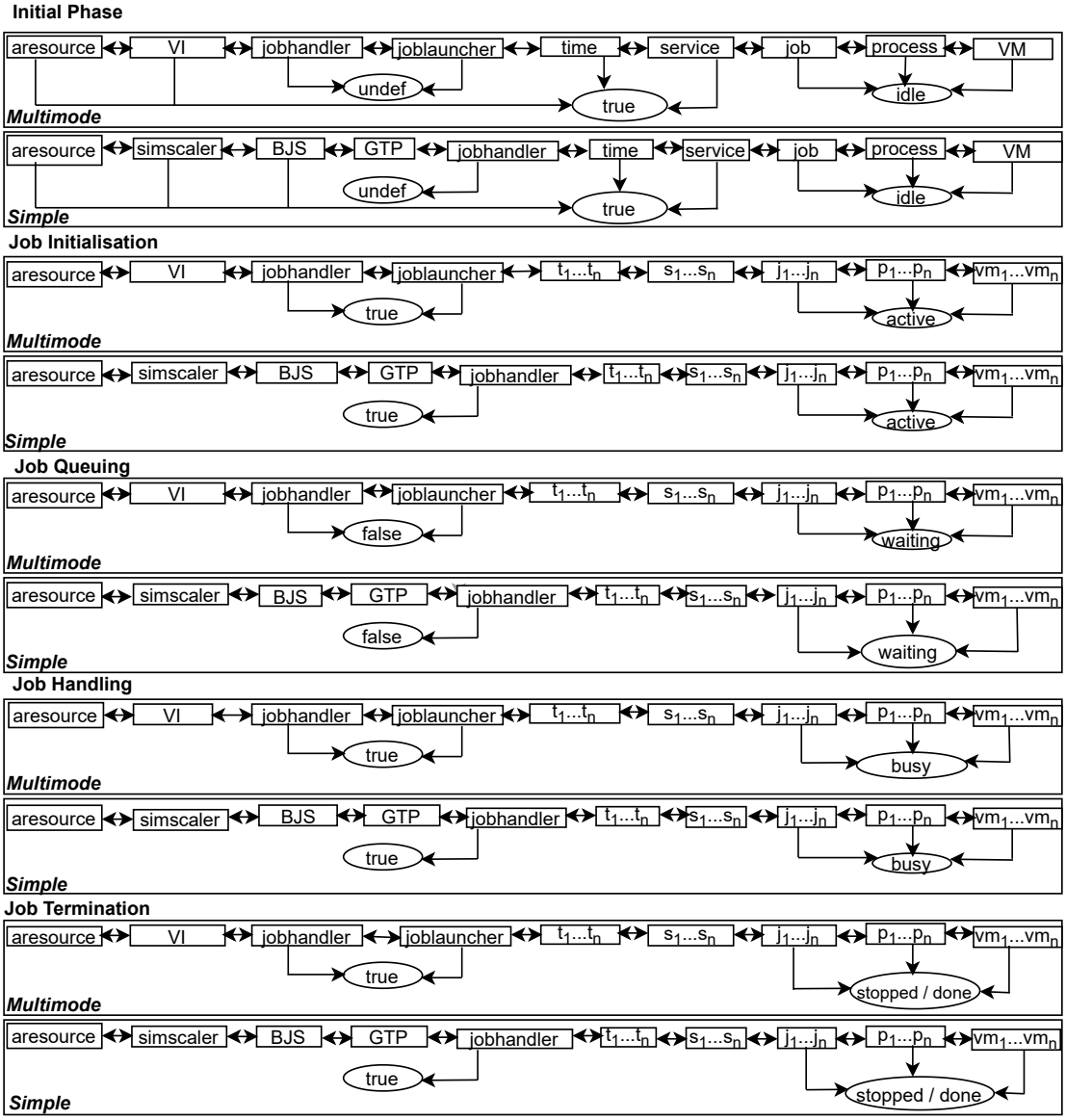


Figure 5.5: Ground Model for ASM Auto-Scaling

Table 5.1: List of ASM Functions

JobState: Job → {*idle*, *submitted*, *waiting*, *running*, $failed$, *done*}
JobTime: Time → {*idle*, *started*, *processing*, *stopped*, *completed*}
ProcessState: Process → {*new*, *ready*, *waiting*, *running*, *stopped*}
SystemRequest: Request × AResource → { $true$, $false$}
SystemState: InfraState → {*idle*, *active*, *waiting*, *busy*, *stopped*, *done*}
JobOutcome: Job → { *success*, $failure$}
Compatible: Select(attr(j),attr(vm)) → {$undef$, $true$, $false$}
AddVM: VM × Job → {$undef$, $true$, $false$}
MappedJob: Job × VM → {$undef$, $true$, $false$}
MappedVM: Job × Process → {$undef$, $true$, $false$}
ReqResources: SystemReq. × AResource → {$undef$, $true$, $false$}
JobRequest: Job × AResource → {$undef$, $true$, $false$}
ProcessRequest:Process × AResource → {$undef$, $true$, $false$}
Event: Task → { *start*, *aborted*, *terminated*}
InitReslist: $IReslist$→ { IRL_{active}, IRL_{idle}, IRL_{busy}}
QueReslist: $QRlist$→ { QRL_{active}, QRL_{idle},QRL_{busy}}
JobHandReslist: $JobhReslist$→ { $JHRL_{active}$, $JHRL_{idle}$, $JHRL_{busy}$}
InitReqFunctions: $InitReqFun$→ { IRF_{active}, IRF_{idle}, IRF_{busy}}
JobProcessing: $Jobproc$→ { JP_{active}, JP_{idle}, JP_{busy}}
Job: Process → Job
Jobhandler: Job → Joblauncher, Job → VM
Submitted: Job × VM → {$undef$, $true$, $false$}
BelongsTo: AResource × VM → {$undef$, $true$, $false$}
DestroyVm: VM → {$undef$, $true$, $false$}
ThresholdLevel: TLevel → { $undef$, T_{min}, T_{avg}, T_{max}}
VmPosition: VMPost → {$undef$, VM_F, VM_L}
ReusableVm: RVM → {$undef$, Q_{min}, Q_{avg}, Q_{max}}
VmPool: Vmpool→ {$undef$, Q_{min}, Q_{avg}, Q_{max}}
NumofServiceRequest: NumofSerReq→ { Num_{min}, Num_{avg}, Num_{max}}
VMCount: NumofVMs→ { Num_{min}, Num_{avg}, Num_{max}}
VmUtilLevel: VmUL → {$undef$, $UtVM_{min}$, $UtVM_{avg}$, $UtVM_{max}$}
SimulationDuration: SimDur→ { SD_{min}, SD_{avg}, SD_{max}}
AveragQueTime: AvgQT → { AQT_{min}, AQT_{avg}, AQT_{max}}
AveragUtilPM: AvgUPM → { $AUPM_{min}$, $AUPM_{avg}$, $AUPM_{max}$}
WorkloadPrediction: PredWorkload → { PWL_{active}, $PWL_{inactive}$}

figure 5.3 as explained above. The transitions occur per the application of an ASM run during task processing.

JobTime depicts periods reflective of job states during task processing. It combines with *JobState* to describe at what period a particular state change occurred.

ProcessState illustrates process state transitions from new to stopped during job processing as shown in figure 5.4 as explained above.

JobOutcome shows the results of an ASM run. It shows either success or failure after job processing.

JobRequest invokes jobs generation which are mapped to VMs for task processing. It combines with *ProcessRequest* to maintain a job and process requests during job initialising and handling.

ProcessRequest invokes the provision of processes and maps them to tasks for task processing.

MappedVM monitors the state of VMs and jobs connection for job initialising and handling. It combines with *MappedJob* to maintain the link between jobs and VMs.

BelongsTo This function ensures that there is enough aresources to support a VM before it is selected.

Compatible ensures that the VM selected is the appropriate one. This is done to prevent the selection of VMs with less utilisation which can be destroy within a short period.

AddVM attaches a VM to a job at the VMs selection stage of job queuing. It combines with *Compatible* and *BelongsTo* to foster the VM selection process.

DestroyVM is activated to destroy VMs when the need arises. It is utilised by auto-scalers to remove unused VMs. additionaly, it is used by certain auto-scalers to monitor the duration VM utilisation. When the VMs exceed those durations, *DestroyVM* is activated to remove them.

NumofServiceRequest monitors the number of service requests made during job processing. It determines whether to scale up or scale down resource provisions in workload prediction auto-scaling.

SystemRequest is the refinement for ProcessRequest and JobRequest.

ReqResources is the refinement for MappedVM and MappedJob.

JobProcessing is the refinement function for the provision and monitoring the vital portion of the job handling. It ensures that sufficient time is allocated for tasks. It also models the outputs of job processing.

VMCount monitors the number of VMs provisioned for task processing. If the number of VMs available are below the expected threshold, a situation of VMs shortage is created. This causes jobs to queue. This function is applied in the queuing phase of our model.

InitReslist is a refinement for the provision and monitoring of the aresources, universes and functions required for the first phase (initial phase) of our model.

QueReslist is a refinement for the provision and monitoring of the aresources, universes and functions required for the third phase (job queuing) of our model.

Systemstate is a the refinement for *ProcessState*, *JobState*, and *JobTime* to reflect system state changes.

VmRequest is the refinement for the VM selection process during job queuing.

$InitReqFunctions$ is the refinement for the provision and monitoring of the aresources, universes and functions required for the second phase (job initialising) of our model.

JobHandReslist is the refinement for the provision and monitoring of the aresources, universes and functions required for the fourth phase (job handling) of our model. It combines with a derived function called *jobhandling module* to process the jobs generated.

Auto-scaler design is focused on optimizing metrics about the virtual infrastructure. We list functions modelling these metrics.

TLevel defines the VMs threshold required for certain auto-scalers. The threshold could be T_{min}, T_{avg}, or T_{max} for minimum, average and maximum thresholds respectively.

$VmUL$ defines VM utilization levels during job processing. The VM utilization levels could be $VMut_{min}$, $VMut_{avg}$ or $VMut_{max}$.

Table 5.2: ASM Verification Notations

Function Notations	**State Trans.:=True**	**State Trans.:=False or Undef**
SystemRequest==sr	srt	srf
MappedJob==mj	mjt	mjf
MappedVM==mv	mvt	mvf
ReqResources==rr	rrt	rrf
JobRequest==jr	jrt	jrf
ProcessRequest==pr	prt	prf
Jobhandler==jh	jht	jhf / jfu
JobLauncher==jl	jlt	jlf / jlu

VmPool defines VM provisions in VM pools. Also, *VmPool* implements *RVM* to monitor VM optimisation levels for reusable VMs. The quantity of VMs in the pool could be Q_{min}, Q_{avg} or Q_{max} for minimum, average and maximum quantities respectively.

VMPost defines VMs' position in the VI during job processing. VMs positions could be VM_F, VM_L for first and last positions which depicts the particular virtual machine that is being monitored by certain auto-scalers. These auto-scalers destroy VMs with less utilisation, unless the VM is the last one to be processed. If it is the last VM, its processing is extended for an hour before it is removed.

These *functions* and *universes* were combined to create the algorithms in the model.

5.3.3 Design of Astam Verification Process

In order for our *Astam* model to be applicable, our ground model and its refinement must be correct and equivalent to each other. To achieve this, the CTL connectives discussed in section 2.10.1.4 will be employed. Moreover, this design process forms part of step 1 of our model's design; which will be applied in step 4 for our model's verification.

The verification will be carried out in two stages. The phases of our ground model and their refinements will be verified during an ASM run. To achieve this verification, CTL connective were applied on the main ASM universes and functions to enable state transitions.

Algorithm 5 *Simple Initial Phase*

1: **if** $\exists vm \in VM \wedge \exists p \in PROCESS \wedge \exists ar \in ARESOURCE \wedge \exists j \in JOB \wedge \exists t \in Time$ **then**
2: $processState(p) := idle$
3: **end if**
4: **while** $SimScaler(j, vm) \wedge BJS(j, vm) \wedge GTP(j)$ **do**
5: **if** $jobRequest(j, ar) = true \wedge processRequest(p, ar) = true$ **then**
6: $JobTime(j) := idle$
7: **end if**
8: **if** $mappedVM(j, p) = false \wedge mappedJob(j, p) = false$ **then**
9: $JobTime(j) := idle$
10: **end if**
11: **if** $installed(j, vm) = false \wedge Jobhandler(j, vm) = undef$ **then**
12: $JobState(j) := idle \wedge JobTime(j) := idle$
13: **end if**
14: **end while**

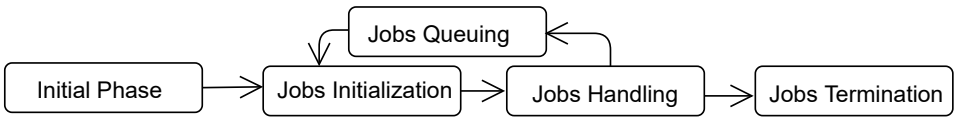


Figure 5.6: ASM Modelled Auto-Scaling Phases

The functions are *jobhandler*, *joblauncher*, *jobRequest*, *processRequest*, *MappedVM*, *MappedJobs*, *SystemRequest* and *ReqResources*.

The *jobRequest*, *processRequest*, *MappedVM* and *MappedJobs* are used for our ground model while *SystemRequest* and *ReqResources* are utilised for the refinements.

In order to ensure flexibility in the verification process, verification notations are derived from the previous universes and functions as seen in table 5.2 on page 117. Table 5.2 shows the possible state changes for our verification notation.

5.4 Refinement of the Multimode and Simple Mechanisms

In this section we will discuss steps 2 and 3 of ASM model to ensure model coherence.

Astam comprises of five *Transition Rules* as seen in figure 5.6. These

Algorithm 6 *Multimode Initial Phase*

1: **if** $\exists vm \in VM \wedge \exists p \in PROCESS \wedge \exists ar \in ARESOURCE \wedge \exists j \in JOB \wedge \exists t \in Time$ **then**
2: $processState(p) := idle$
3: **end if**
4: **while** $VI(j, vm) \wedge Joblauncher(Jobhandler(j, vm))$ **do**
5: **if** $jobRequest(j, ar) = false \wedge processRequest(p, ar) = false$ **then**
6: $JobTime(j) := idle$
7: **end if**
8: **if** $mappedVM(j, p) = false \wedge mappedJob(j, p) = false$ **then**
9: $JobTime(j) := idle$
10: **end if**
11: **if** $installed(j, vm) = false \wedge Joblauncher(Jobhandler(j, vm), vm) = false$ **then**
12: $JobState(j) := idle \wedge JobTime(j) := idle$
13: $numofSerReq(s_i) := undef \wedge VmUL = undef$
14: $Vmpool := undef \wedge RVM := undef$
15: **end if**
16: **end while**

rules are designed to reflect the execution phases an auto-scaler undergoes during job processing. The rules enable users to analyse the VM provision behaviours of auto-scaling mechanisms. We utilised algorithms to express the details of our *ground model* shown in figure 5.5. These algorithms were further refined according to the ASM refinement method (in definition 3) into lower levels of abstractions.

The VM provision behaviours of our ground model and the refinements are later compared for equivalence according to *Börger's refinement* (described in definition 4) to check for the consistency of state transitions.

Astam's Transition Rules are: (*i*) Initial Phase (*ii*) Job Initialising (*iii*) Job Queuing (*iv*) Job Handling and (*v*) Job Termination. The initial phase is the first transition rule of our model. This is the phase where all requisite resources are availed for job processing to commence. The system state is *idle* in this phase. The job initialising phase is the stage where job processing commences with the activation of system requests, and the mapping of jobs to VMs. The system state transitions to *active* in this phase. The job queuing phase is the stage where jobs queue due to the unavailability of VMs. The system state transitions to *waiting* in this phase. The job handling stages is the actual job processing stage where all the requisite resources are availed and jobs are allowed to processed till completion. The system state

Algorithm 7 *Refined Initial Phase*

Require: AResource
1: **if** $InitReslist = IRL_{idle} \wedge ReqResources = false$ **then**
2: $SystemState(j, p) := idle$
3: **end if**
4: **while** $SystemRequest = false \wedge ReqResources = false$ **do**
5: $SystemState(j, p) := idle$
6: **end while**
7: **if** $installed(j, vm) = false \wedge Jobhandler(j, vm) = undef$ **then**
8: $SystemState(j, p) := idle$
9: **end if**

transitions to *busy* in this phase. The job termination phase is the final phase of our transition rules. This is the stage where all upload jobs are completely processed or a system interrupt causes job processing to halt. The system state transitions to either *done* or *stopped* in this phase.

The *rules* are described below in conjunction with algorithms 5 to 19 to identify the auto-scaling common practices. *Astam* is applied to auto-scalers from multiple sources in algorithms 20 to 34 to evaluate its applicability.

Moreover, derived functions (ASM modules) were developed and applied to the job execution phases of our model. These modules were introduced to ensure modularisation. In the next sub-sections, we shall discuss our *Astam's Transition Rules*.

5.4.1 Astam's Transition Rules

Astam's rules are described below from initial phase to job termination. State transitions are highlighted to reflect the state changes in during job processing. Also, we compare the two categories of auto-scalers to identify similarities in their auto-scaling behaviours. Now let us discuss our first transition rule.

5.4.1.1 Rule 1, Initial Phase

Algorithms 5 and 6 depict the first phase of our model in figure 5.5. They also represent the initial phase for both *Simple* and *Multimode* auto-scalers.

At the initial phase, all requisite universes are provisioned for job processing to commence, however the process state is updated to *idle* as seen in lines 1 to 2 of algorithms 5 and 6. The system is inactive due to the absence of the ASM rule that causes job processing to commence.

Algorithm 8 *Simple Jobs Initialising*

1: **if** $\exists vm \in VM \wedge \exists p \in PROCESS \wedge \exists ar \in ARESOURCE \wedge \exists j \in JOB \wedge \exists t \in Time$ **then**
2: $processState(p) := ready$
3: **end if**
4: **while** $SimScaler(j, vm) \wedge BJS(j, vm) \wedge GTP(j)$ **do**
5: **if** $jobRequest(j, ar) = true \wedge processRequest(p, ar) = true$ **then**
6: $JobTime(j) := started$
7: **end if**
8: **if** $mappedVM(j, p) = true \wedge mappedJob(j, p) = true$ **then**
9: $JobTime(j) := started$
10: **end if**
11: **if** $installed(j, vm) = true \wedge Jobhandler(j, vm) = true$ **then**
12: $JobState(j) := submitted$
13: **end if**
14: **end while**

JobS and VMs provisions are monitor by both auto-scalers. The *Simple* auto-scaler utilise the SimScaler(j,vm), BJS(j,vm) and GTP(j), while the *Multimode* auto-scaler applies the VI(j, vm) and the joblauncher to do same as seen in lines 4. Also, jobs and process requests are made but no response is receieved. Jobs are not submitted to VMs, as such the VMs remain unmapped until an ASM rule is applied to cause a system state change to occur.

The *Mutimodes* activate functions specific to their behaviours as seen in lines 13 to 14 of algorithm 6. These functions are activated to monitor specific indicators during auto-scaling. However, since job processing is not initialised, they are all updated to *idle*. Algorithms 5 and 6 are refined in algorithm 7.

Initial Phase Refinement The *InitReslist* derived function is utilised to check for the provision of requisite universes for this phase. *InitReslist* is a refinement for all required universes and functions for the initial phase. *systemstate* is updated to *idle* as seen in lines 2. Since there are not activities at this stage. *Systemrequests* and *ReqResources* are mapped to *false* as jobs are not submitted to VMs. Also, Jobs and VMs are not installed as tasks for processing for by the jobhandler for processing. This causes *systemstate* to remain updated to *idle*.

This refinement is equivalent to our ground model's algorithms discussed in algorithms 5 and 6, and seen in figure 5.5. Since the state

Algorithm 9 *Multimode Jobs Initialising*

1: **if** $\exists vm \in VM \wedge \exists p \in PROCESS \wedge \exists ar \in ARESOURCE \wedge \exists j \in JOB \wedge \exists t \in Time$ **then**
2: $processState(p) := ready$
3: **end if**
4: **while** $VI(j, vm) \wedge Joblauncher(Jobhandler(j, vm))$ **do**
5: **if** $jobRequest(j, ar) = true \wedge processRequest(p, ar) = true$ **then**
6: $JobTime(j) := started$
7: **end if**
8: **if** $mappedVM(j, p) = true \wedge mappedJob(j, p) = true$ **then**
9: $JobTime(j) := started$
10: **end if**
11: **if** $installed(j, vm) = true$ **then**
12: $JobState(j) := submitted$
13: **end if**
14: **end while**

changes of algorithm 7 are equivalent to the state transitions of our ground model. This is seen in table 5.1, where the ground model functions can be refined to derived functions and vice-versa.

5.4.1.2 Rule 2, Job Initializing

The second phase of our model (shown in figure 5.5) begins with a system call (which is activated by the application of an *control state ASM rule* during an ASM run). The system call provisions the universes assigned from the

Algorithm 10 *Refined Job Initialising*

Require: AResource
1: **if** $InitReqFunctions = IRF_{active}$ **then**
2: $SystemState(j, p) := active$
3: **while** $SystemRequest = true \wedge ReqResources = true$ **do**
4: $SystemState(j, p) := active$
5: **end while**
6: **if** $installed(j, vm) = true \wedge Jobhandler(j, vm) = true$ **then**
7: $SystemState(j, p) := active$
8: **end if**
9: **end if**

Algorithm 11 *Simple Jobs Queuing*

1: **if** $\exists j \in JOB \wedge \exists ar \in ARESOURCE \wedge \exists p \in PROCESS \wedge \exists vm \in VM \wedge \exists t \in Time$ **then**
2: $JobTime(j) := started$
3: **end if**
4: **while** $SimScaler(j, vm) \wedge BJS(j, vm) \wedge GTP(j)$ **do**
5: $processState(p) := ready$
6: **if** $jobRequest(j, ar) = true \wedge processRequest(p, ar) = true$ **then**
7: $JobTime(j) := started$
8: **end if**
9: **if** $VMCount \leqslant Num_{min}$ **then**
10: $MappedVM(j, vm) := false \wedge Jobhandler(j, vm) := false$
11: $JobState(j) := waiting$
12: **end if**
13: **end while**

previous phase. This transitions *processState* from *new* to *ready* as seen in lines 2 of our ground model algorithms 8 and 9 .

The auto-scalers specific universes are provisioned to monitor the activities of jobs and VM activities in line 4. The *Simple* auto-scalers utilise SimScaler(j,vm), BJS(j,vm) and GTP(j) while *Multimode* auto-scalers apply VI(j,vm) and the joblauncher.

JobRequests and *processRequests* are activated to connect VMs to Jobs. This transitions *jobtime* to *started* as seen in line 6. Also, the system call activates mapped VMs and jobs, which causes jobs and VMs to be installed as *tasks* as seen in lines 8 to 11. The *jobhandler* is activated to process the *tasks* in *Simple* scalers and *jobhandler* in *Multimode* auto-scalers.

The *JobTime* and *Jobstate* are updated to*started* and *submitted* respectively as seen from lines 12.

The refinement of algorithms 8 and 9 are modelled in algorithm 10.

Job Initialising Refinement The *InitReqFunctions* ASM derived function is introduced to check the provisioning of requisite universes for this phase as seen in line 1 of algorithm 10. *InitReqFunctions* is a refinement for all required universes and functions for job initialising.

The authentication of the universes updates *Systemstate* to *active*. *SystemRequest* is activated for the job requests and VMs provisions. This causes *systemstate* to be updated to *active* as seen in line 3 to 4.

ReqRequest is applied to map jobs to VMs to which are installed as tasks for the Jobhandler and Joblauncher to enforce their processing.

Algorithm 12 *Multimode Jobs Queuing*

1: **if** $\exists j \in JOB \wedge \exists ar \in ARESOURCE \wedge \exists p \in PROCESS \wedge \exists vm \in VM \wedge \exists t \in Time$ **then**
2: $JobTime(j) := started$
3: **end if**
4: **while** $VI(j, vm)$ **do**
5: $processState(p) := ready$
6: **if** $jobRequest(j, ar) = true \wedge processRequest(p, ar) = true$ **then**
7: $JobTime(j) := started$
8: **end if**
9: **if** $VMCount \leqslant Num_{min}$ **then**
10: $MappedVM(j, vm) := false$
11: $Joblauncher(Jobhandler(j, vm)) := false$
12: $JobState(j) := waiting$
13: **end if**
14: **end while**

The activities of these functions, cause the *Systemstate* to be updated to active as seen in line 7.

This refinement is equivalent to the algorithms 8 and 9 and the job initialising phase of our ground model as seen in figure 5.5. Since the state changes of algorithm 10 are equivalent to the state transitions of our the ground model after the ASM run. This is seen in table 5.1, where the ground functions can be refined to derived functions and vice-versa.

5.4.1.3 Rule 3, Job Queuing

The job queuing phase commences when there is a shortage of VMs during job processing. The phase is modelled as part of our ground model is seen in algorithms 11-12 and figure 5.5 for *Simple* and *Multimode* auto-scalers.

When job queuing begins, universes are provisioned as part of the resources from the previous phases. This causes *jobTime* and *processState* to transitioned to *started* and *ready* respectively as seen in lines 1 to 5.

Job and process requests are activated to ensure VM provisions as seen in lines 4 to 7. The VM count is monitored during job processing to determine the quantity available. If the quantity is below the required threshold for job processing; The *MappedVM*, *Jobhandler* and *joblauncher* are updated to *false* to confirm low VM count. This causes the jobs provisioned to queue as there

Algorithm 13 *Refined Jobs Queuing*

Require: AResource

1: **while** $InitReqFunctions = IRF_{active} \wedge ReqResources = true$ **do**
2: $SystemState(j, p) := active$
3: **if** $QueReslist = QRL_{active} \wedge SystemRequest(p, ar) = true$ **then**
4: $SystemState(j, p) := active$
5: **if** $VMCount \leqslant Num_{min}$ **then**
6: $ReqResources := false \wedge Jobhandler((j, vm) := false$
7: $SystemState(j, p) := waiting$
8: **end if**
9: **end if**
10: **end while**

is a shortage of VMs. This is seen in lines 9 to 12 of both algorithms. *JobState* transitions to *waiting* to signify the current state of the modelling process. Our ground model algorithms are refined in algorithm 13.

Job Queuing Refinement The *InitReqFunctions* and *ReqRequest* functions are activated to foster resource provision and the mapping of VMs to jobs. This causes *SystemState* to be updated to *active* as seen in lines 1 to 2. The *QueReslist* derived function is introduced to provision universes and functions for job queuing. Also, *SystemRequest* is activated to initiate jobs and VMs requests. These activities maintains *systemstate* at *active* as seen in lines 3 to 4.

The VM count is monitored to check for the quantity of available VMs. A reduction in the VM count causes *ReqRequest* and *Jobhandler* to be updated to *false*. This situation causes the *SystemState* to transition to *waiting* as seen in lines 5 to 7.

This refinement of job queuing is equivalent to our ground model algorithms, as they all have their system states transitioning to *waiting* as seen in figure 5.5. This is seen in table 5.1, where the ground functions can be refined to derived functions and vice-versa.

5.4.1.4 Rule 4, Job Handling

The Job handling is the fourth phase of our model. In order to model the job handling stage, there is the need to apply an a derived function called the *Jobhandling Module* to optimize job handling. The purpose of the *Jobhandling Module* is to optimise the VM selection process during the job queuing

Algorithm 14 *Simple Jobhandling Module* $JobHandMod_{simple}$

Require: AResource

1: **while** $InitReqFunctions = IRF_{active}$ **do**
2: $SystemState(j, p) := active$
3: **if** $QueReslist = QRL_{active} \wedge SystemRequest(p, ar) = true$ **then**
4: $SystemState(j, p) := active$
5: **end if**
6: **while** $SimScaler(j, vm) \wedge BJS(j, vm) \wedge GTP(j)$ **do**
7: **if** $VMCount \leqslant Num_{min}$ **then**
8: $ReqResources := false \wedge Jobhandler((j, vm) := false$
9: $SystemState(j, p) := waiting$
10: **end if**
11: **if** $(ReqResources = false \wedge installed(task(j, vm)) = false$ **then**
12: $AddVM(vm, j) := true \wedge Compatible(attr(j), attr(vm)) := true$
13: $belongsTo(j, vm) := true$
14: $ReqResources := true$
15: **end if**
16: **end while**
17: **end while**

phase. This function is created for each category of auto-scalers (i.e. *Simple* and *Multimode Jobhandling* module). The *Simple* and *Multimode Jobhandling* Modules shown in algorithms 14 and 15 perform similar functions but with structural differences.

The *Simple Jobhandling* Module utilise *InitReqFunctions* and *QueReslist* to check for the provision for job initialising and queuing universes and functions. Once they are authenticated, *systemState* is updated to *active* as seen in lines1 to 4.

Auto-scaler specific universes are provisioned to foster the exhibitions of varied VM provision behaviours during job processing. This activates VMs and jobs requests. The VM count is monitored for expected quantities. If the VM count is below the required threshold, a state change occurs. This causes *ReqResources* (which maps jobs to VMs) to be updated to *false*, as well as the *Jobhandler*. The *systemState* transitions to *waiting* as seen in lines 6 to 10 of algorithms 14 and 15.

The *ReqResources* is rechecked periodically, to confirm if jobs have been mapped to VMs and installed as tasks. If the response is negative, the VM selection mode is activated via *AddVM* function. The VM selections process, checks if the appropriate VMs (i.e. VMs with the required level of utilisation) are being selected. Once these features have been confirmed to be available,

Algorithm 15 *Multimode Jobhandling Module* ($JobHandMod_{multimode}$)

Require: AResource

1: **while** $InitReqFunctions = IRF_{active}$ **do**
2: $SystemState(j, p) := active$
3: **if** $QueReslist = QRL_{active} \wedge SystemRequest(p, ar) = true$ **then**
4: $SystemState(j, p) := active$
5: **end if**
6: **while** $VI(j, vm)$ **do**
7: **if** $VMCount \leqslant Num_{min}$ **then**
8: $ReqResources := false \wedge Joblauncher(Jobhandler(j, vm), vm) := false$
9: $SystemState(j, p) := waiting$
10: **end if**
11: **while** $VmRequest(j, vm) = active$ **do**
12: $ReqResources := true$
13: **end while**
14: **end while**
15: **end while**

the jobs are then mapped to VMs as seen in lines 11 to 16 of algorithm 14.

The VM selection process is slightly different for the *Multimode* Jobhandling module. The *VmRequest* function (which is a refinement of the VM selection process) is activated. This causes jobs to be mapped to VMs via the *ReqResources* as seen in lines 11 to 14 of algorithm 15. The application of the *Jobhandling Modules* fosters varied VM provision modelling behaviours in the categories of auto-scalers.

Job handling commences with the application of *InitReqFunctions* to check the provision of job initialising universes and functions. Once these are *active*, the *JobhandMod* is activated to provision all the universes and functions for job handling. This causes *SystemState* to be updated to *active* as seen in lines 1 to 3 of algorithm 16.

Sufficient time request is made and the response granted by the *AResources* to ensure that the jobs provisioned are adequately processed. The *SystemRequest* is actovated to foster jobs and process requests. An authentication of this request, maps jobs to VMs via *ReqResources*. This causes the mapped jobs and VMs to be installs as tasks to either continue (i.e. if the jobs were queuing) or commence job handling (i.e. if the job processing had just initialised) job processing activity as seen in lines 5 to 7.

The outputs of job processing are monitored with *SimulionDuration*, *Av-*

Algorithm 16 *Job Handling*

Require: AResource

1: **while** $InitReqFunctions = IRF_{active} \wedge SystemRequest(p, ar) := true$ **do**
2: $JobHandMod_{multimode} := active \wedge Jobhandler((j, vm) := true$
3: $SystemState(j, p) := active$
4: **while** $t \in TIME \wedge TimeRequest(j, p) = true \wedge mappedVM(j, vm) = true$ **do**
5: **if** $SystemRequest(p, ar) = true$ **then**
6: $ReqResources := true$
7: $installed(j, vm) := true \wedge event(t) = started$
8: $SimulationDuration := SD_{max}$
9: $AveragUtilPM := AUPM_{max}$
10: $AveragQueTime := AQT_{max}$
11: **end if**
12: $SystemState(j, p) := busy$
13: **end while**
14: **end while**

erageUtilPM and *AverageQueTime*. These three functions monitor the period utilised for job processing, the average utilisation of PMs and the average queuing time of the VMs. The job processing activity causes the *SystemState* to transition to *busy* as seen in lines 9 to 12.

Our ground model's job handling algorithms are refined in algorithm 17.

Job Handling Refinement In order to ensure that the requisite universes for job initialising are provisioned, the *InitReqFunctions* and *SystemRequest* functions are activated. This causes *SystemState* to be updated to *active* as seen in line 1 of algorithm 17. Also, the *Jobhandling Module* and the *jobhandler* are activated to foster VMs selection during job queuing and the installing of mapped VMs and jobs as tasks for job processing as seen in line 2.

The *JobProcessing* function is activated to foster time requests and the mapping of jobs to VMs for job processing. Also, the output of job handling show are modelled as the job processing ensues. This activity causes the *SystemState* to transition to *busy*as seen in lines 5 to 12.

This refinement is equivalent to our ground model's job handling algorithm, as the ASM run causes *SystemState* to transition to *busy* as seen in figure 5.5. This is seen in table 5.1, where the ground functions can be refined to derived functions and vice-versa.

Algorithm 17 *Refined Job Handling*

Require: AResource

1: **while** $InitReqFunctions = IRF_{active} \wedge SystemRequest(p, ar) := true$ **do**
2: $JobHandMod_{multimode} := active \wedge Jobhandler(j, vm) := true$
3: $SystemState(j, p) := active$
4: **while** $JobProcessing = JP_{active}$ **do**
5: $SystemRequest(p, ar) := true \wedge ReqResources := true$
6: $SystemState(j, p) := busy$
7: **end while**
8: **end while**

5.4.1.5 Rule 5, Job Termination

Job Termination is the fifth phase of our model. This phase requires two conditions to be initiated. First, a system failure or an abrupt system call to halt job processing. Second, the exhaustion of jobs generated (i.e. the complete processing of uploaded jobs).

The job termination phase is as a result of the activities of job handling as seen in algorithm 18 of our ground model. Hence, there must be ongoing activities to demonstrate job processing in the system, before job termination occurred.

Therefore, before the processing of jobs is terminated, the universes and functions required for job initialising should be provisioned. This updates *processState* to *ready* as seen in lines 1 to 2.

Job processing is monitored via amount of time allocated, and how long jobs remain mapped to VMs. As job are being processed, *ProcessState* and *jobState* transtion to *running*. Also *jobTimes* updated to *processing* as seen in lines 3 to 10. When a system interrupt occurs which causes the job processing to halt; *jobstate* transitions to *failed* and *processState* to *stopped* as seen in lines 11 to 13.

Moreover, when job processing is completed; *jobState* transitions to *done*, *jobTime* to *completed*, and *event(t)* to *terminate* as seen in lines 15 to 18 depicting the exhaustion of job generated.

The state changes can be seen in the process and job states figures 5.4 and 5.3. The job termination algorithm is refined in algorithm 19.

Job Termination Refinement is accomplished via he introduction of primary derived functions such as *InitReqFunctions*, *SystemRequest* and *ReqResources*. These function ensures that adequate *universes* and *functions* are provisioned for job initialising and job handling. This

Algorithm 18 *Job Termination*

1: **if** $\exists j \in JOB \wedge \exists ar \in ARESOURCE \wedge \exists p \in PROCESS \wedge \exists t \in Time \wedge JobRequest(j, ar)$ **then**
2: $processState := ready$
3: **while** $t \in TIME \wedge TimeRequest(j, p) = true \wedge mappedVM(j, vm) = true$ **do**
4: $JobTime(j) := processing$
5: **while** $processRequest(p, ar) = true \wedge jobhandler(j, vm) = true$ **do**
6: $installed(j, vm) := true \wedge event(t) := started$
7: $JobState(j) := running \wedge ProcessState(p) := running$
8: $JobTime(j) := processing$
9: **end while**
10: **end while**
11: **if** $jobRequest(j, ar) = true \wedge event(task(p)) = terminate$ **then**
12: $JobState(j) := failed \wedge processState(p) := stopped$
13: **end if**
14: **end if**
15: **if** $\nexists j \in JOB \wedge \nexists p \in PROCESS \wedge JobRequest(j, ar)$ **then**
16: $Event(t) := Terminate \wedge jobTime(j) := Completed$
17: $jobState(j) := done$
18: **end if**

causes *systemstate* to transitions to *active* as seen in lines 1 to 2 of algorithm 19.

The *JobHandReslist* function (a refinement of job handling specific functions) and the *jobhandler* are activated to foster job processing. This causes *systemstate* to transition to *busy* as seen in lines 3 to 4.

When a system call arrives that causes job processing to terminate while *systemstate* is *busy*; the *systemstate* is automatically updated to *stopped* as seen in lines 5 to 6. This event signifies an abrupt job termination.

Furthermore, when *SystemRequest* is *active* while there are not jobs for VMs to process; *event* is updated to *terminate* and *systemstate* to *done* as seen in lines 7 to 9. This signifies the completion of job processing.

This refinement (algorithm 19) is equivalent to our ground model's algorithm 18 as *systemstate* transitioned to either *stopped* or *done* as a response to the job termination conditions. This is seen in table 5.1, where the ASM functions can be refined to derived functions and vice-versa. Also, it is reflective of this phase depicted in our ground model

Algorithm 19 *Refined Job Termination*

Require: AResource

1: **while** $InitReqFunctions = IRF_{active} \wedge SystemRequest(p, ar) = true$ **do**
2: $ReqResources := true \wedge SystemState(j, p) := active$
3: **while** $JobHandReslist = JHRL_{active} \wedge jobhandler(j, vm) = true$ **do**
4: $SystemState(j, p) := busy$
5: **if** $SystemState(j, p) = busy \wedge event(task(p)) = terminate$ **then**
6: $SystemState(j, p) := stopped$
7: **else if** $SystemRequest(p, ar) \wedge \nexists j \in JOB \wedge \nexists p \in PROCESS$ **then**
8: $Event(t) := Terminate$
9: $SystemState(j, p) := done$
10: **end if**
11: **end while**
12: **end while**

figure in 5.5 and our modelled auto-scaling phases figure in 5.6

5.5 Summary

In this chapter, we designed our *Astam* model via investigating of the resource provision behaviours of auto-scalers offered by the DISSECT-CF simulator. The investigations helped us to comprehend how the various components of the simulator are connected. Two categories of auto-scalers (*simple* and *multimode*) were identified with their specific features. It was observed that, the *VMs* provisioning is done per the specifications of respective mechanisms. Some auto-scalers prefer VM Pools, while others create VMs during auto-scaling. This is accomplished while monitoring the *utilization levels* of VMs. These observations helped us to created extra auto-scalers and to design our model.

Five Transition rules and a ground model framework were proposed. Algorithms were generated and formalized from the auto-scalers identified from our investigations. These algorithms reflected our ground model's framework. These algorithms were refined according to ASM model refinement method discussed in sub-section 2.10.1.2.

The formalized algorithms of the *simple* and *multimode* auto-scalers were compared with each other. Also, the ground models were compared with their refinements. The comparisons were accomplished via *Astam's* transitions rules and in accordance with *Börger's refinement* discussed in definition 4.

Equivalence of state transitions were achieved via the application of derived functions and control ASMs. As such, refined algorithms can be refined back to their ground models where required.

The comparisons of the two categories of auto-scalers enabled us to conclude that, although the auto-scalers were developed on different frameworks, they exhibited similar VM provision behaviours during job processing.

Chapter 6
Astam Evaluation, Verification and Validation

6.1 Introduction

In this chapter, we will discuss the evaluation, verification and validation of our *Astam* model. The evaluation describes the application of our model to formalized algorithms of specific auto-scalers offered by DISSECT-CF. Also, we describe the application of *Astam* to other auto-scalers whose algorithms have been made public. The verification section focuses on the application of computational tree logic properties to determine the correctness of *Astam*. Then, we validate *Astam* by generating test cases from the formalized algorithms and running them on CoreASM toolkit to determine the adherence of our model to *guarded updates*. Now let us discuss the evaluation of *Astam*.

6.2 Astam Model Evaluation

This section describes the observed state transitions and the equivalence of our ground model and its refinements, when *Astam* was applied to specific auto-scalers. Our evaluation criteria are discussed into detail to emphasize the application of ASM refinement method described in sub-section 2.10.1.

6.2.1 Discussion of Astam Evaluation criteria

We applied the following criteria to evaluate our *Astam* model. Let us now focus on discussing the criteria.

- To assess the equivalence of the refined auto-scaler algorithms to the ground model via the application of *universes* and *signatures*. This

Algorithm 20 *Threshold Initial Phase*

Require: AResource

1: **if** $InitReslist = IRL_{idle}$ **then**
2: $TLevel := undef \wedge SystemState(j, p) := idle$
3: **end if**
4: **while** $SystemRequest = false$ **do**
5: $SystemState(j, p) := idle$
6: **end while**
7: **if** $ReqResources = false$ **then**
8: $SystemState(j, p) := idle$
9: **end if**
10: **if** $installed(j, vm) = false \wedge Joblauncher(Jobhandler(j, vm)) = false$ **then**
11: $SystemState(j, p) := idle$
12: **end if**

Algorithm 21 *Vmopt Jobs Initializing*

Require: AResource

1: **if** $InitReqFunctions = IRF_{active} \wedge SystemRequest = true$ **then**
2: $Joblauncher(Jobhandler(j, vm)) := true \wedge ReqResources := true$
3: $RVM := Q_{min}$
4: $SystemState(j, p) := active$
5: **end if**

involves examining the algorithms of the job execution phases of the auto-scalers to ensure that they are equivalent to our ground model.

- To examine the application of derived functions (modules) to portions of auto-scaling modelling; such as job initialising, VM selection and job handling to ensure the application of *ASM Model Refinement*.
- To assess the application of *guarded updates* (which is reflective of control state ASMs) and *Börger's refinement* on auto-scalers.

The functions and state transitions for the initial phase and job initialising are the same for all auto-scalers. Therefore, only one auto-scaler will be modelled to discuss the first two phases. This is to reduce the repetition of algorithms. Let us discuss the evaluation of our ASM rules.

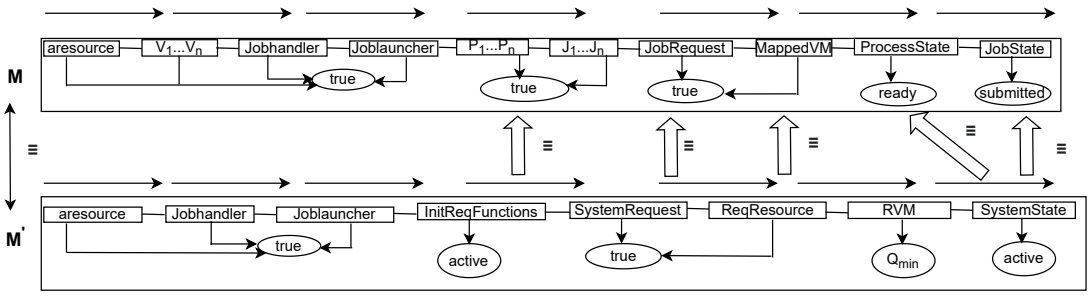


Figure 6.1: Vmopt auto-scaler job initialising

6.2.2 Evaluating Astam's Transition Rules

This sub-section describes the evaluation of our model's rules introduced in 5.4.1. Let us begin the discussions with Rule 1.

6.2.2.1 Rule 1, Initial Phase:

This phase was evaluated per sub-section 5.4.1.1 of our model. The auto-scalers are expected to exhibited state transitions equivalent to the first phase of our ground model as seen in algorithm 5, and its refinement.

Algorithm 20 is used to discuss the evaluation of the intial phase for all auto-scalers, since aside the specific function *TLevel*, all the state changes are the same for all auto-scalers.

At the initial phase, all the auto-scalers apply the *InitReslist* function to access universes and functions. However, no *aresources* are provisioned. Therefore *systemstate* is updated to *TLevel* and the auto-scaler threshold monitoring function *TLevel* is updated to *undef* as seen in lines 1 to 3 of algorithm 20.

Also, job and process requests are made but no responses are received (as expected). Hence, there was no state transitions for *systemstate* as seen in lines 4 to 6. The lack of response to job requests showed that, there was no provision of VMs to be mapped to jobs; and no tasks were likewise installed for processing. This causes the *Joblauncher* to be updated to *false* and system state to *idle* as as seen in lines 7 to 12.

This refinement is equivalent to the initial phase of our ground model shown in figure 5.5. Since the system state transitions to *idle* as seen in algorithm 7. Also, the refinement satisfies the evaluation criteria discussed in sub-section 6.2.1 because derived function were applied to at the initial phase of this refinement. Also, state transitions are observed when conditions were met, which are reflective of *guarded updates* of control state ASMs.

Algorithm 22 *Threshold Jobs Queuing*

Require: AResource

1: **while** $InitReqFunctions = IRF_{active}$ **do**
2: $ReqResources := true \wedge TLevel := T_{min}$
3: $SystemState(j, p) := active$
4: **if** $QueReslist = QRL_{active}$ **then**
5: $SystemState(j, p) := active$
6: **end if**
7: **if** $VMCount \leqslant Num_{min}$ **then**
8: $ReqResources := false$
9: $Joblauncher(Jobhandler(j, vm)) := false$
10: $TLevel := T_{avg}$
11: $SystemState(j, p) := waiting$
12: **end if**
13: **end while**

6.2.2.2 Rule 2, Job Initialising

Job Initialising was evaluated with sub-section 5.4.1.2. Algorithm 21 is utilised to discuss the job initialising evaluation for the auto-scalers, since aside the specific reusable VMs function *RVM*, all the state changes are the same for all auto-scalers.

The *InitReqFunctions* derived function is applied to provision *Aresources* via universes and functions for job initialising. The *SystemRequests* and *ReqResources* functions activate jobs and VMs requests and the mapping of VMs to jobs which are installed as tasks. This causes job processing to commence as seen in lines 1 to 2 and in figure 6.1. The reusable VMs function *RVM* is updated to minimum state. This causes *SystemState* to transition to *active* as seen in lines 3 to 5 of algorithm 21.

This evaluation process is applicable to *threshold*, *vmcreate*, *Pooling* and *FixedVM* auto-scalers. In the case of the other *multimode* auto-scalers, *threshold* transitions to T_{min}, *vmcreate* to T_{min} and *pooling* to Q_{min} during job initialising.

This refinement is equivalent to the job initialising of our model shown in figure 5.5 and algorithm 10. Also, the refinement satisfies our evaluation criteria discussed in sub-section 6.2.1 since derived function were applied to cause job initialising. Also, state transitions are seen when conditions were met, which are reflective of *guarded updates* of control state ASMs. Moreover, from figure 6.1, we can see that the algorithms for job initialising can be refined back to the ground model in an ASM run.

Algorithm 23 *Threshold Jobs handling*

Require: AResource

1: **while** $InitReqFunctions = IRF_{active}$ **do**
2: $JobHandMod_{multimode} := active \wedge ReqResources := true$
3: **if** $JobHandReslist = JHRL_{active} \wedge TLevel = T_{max}$ **then**
4: $SystemRequest(p, ar) := true \wedge SystemState(j, p) := busy$
5: **end if**
6: **if** $CurrThreshold_{vm} < T_{avg}$ **then**
7: $DestroyVM := true$
8: **else if** $(CurrThreshold_{vm} < T_{avg}) \wedge (CurrVmPost := VM_L)$ **then**
9: $CurrTime(vm) + T_{3600s} \wedge DestroyVM(vm) := true$
10: $Joblauncher(Jobhandler(j, vm)) := true \wedge ReqResources := true$
11: $SimulationDuration := SD_{max} \wedge AveragUtilPM := AQT_{max}$
12: $SystemState(j, p) := busy$
13: **end if**
14: **end while**

6.2.2.3 Rule 3, Job Queuing:

Job queuing evaluation was achieved via Rule 3 of the model discussed in sub-section 5.4.1.3. Algorithm 22 representing the the job queuing phase of *threshold* is used for our discussion.

The *InitReqFunctions* is activated to cause the provision of *Aresources* via *universes* to foster job initialising. This caused *SystemState* to transition to *active*. *Tlevel* transitions to T_{min} (i.e., minimum VM threshold utilisation) as seen in lines 1 to 3 of algorithms 23.

QueResList is activated to monitor process and job requests, and the mapping of VMs to jobs. The VM counts are also monitored regularly via *VMCount*. A reduction in the number of VMs, causes *ReqResources* to be updated to *false* (signifying VMs shortage). Also, *Tlevel* transitions to T_{avg} (i.e., average VM threshold utilisation). This causes *SystemState* to transition to *waiting* as seen in lines 4 to 10 of algorithms 22.

This evaluation process is applicable to all auto-scalers. In the case of the other *multimode* auto-scalers *Vmcreate* transitioned to T_{avg}, *Pooling* to Q_{avg} and *Vmopt* to Q_{avg} during job queuing.

This refinement is equivalent to the job queuing of our model shown in figure 5.5 and algorithm 13. Also, the refinement satisfies the evaluation criteria discussed in subsection 6.2.1. This is seen in the application of derived functions to model job queuing. Also, state changes are seen when function conditions were met, which are reflective of *guarded updates* of control state

Algorithm 24 *Vmcreate Jobs handling*

Require: AResource

1: **while** $InitReqFunctions = IRF_{active}$ **do**
2: $JobHandMOD_{multimode} := active$
3: **if** $JobHandReslist = JHRL_{active} \wedge TLevel = T_{max}$ **then**
4: $SystemRequest(p, ar) := true \wedge SystemState(j, p) := busy$
5: **end if**
6: **if** $Ut_{max} > T_{min}$ **then**
7: $AddVM(vm) := true \wedge ReqResources := true$
8: **else if** $(Ut_{avg} > T_{min}) \wedge (CurrThreshold_{vm} > T_{min})$ **then**
9: $AddVM := true \wedge ReqResources := true$
10: $Joblauncher(Jobhandler(j, vm)) := true$
11: $SystemState(j, p) := busy$
12: $SimulationDuration := SD_{max} \wedge AveragUtilPM := AUPM_{max}$
13: $AveragQueTime := AQT_{max}$
14: **end if**
15: **end while**

ASMs.

6.2.2.4 Rule 4, Job Handling:

Job handling was evaluated per Rule 4 (discussed in sub-section 5.4.1.4).

The auto-scaling mechanisms applied the job handling modules created for their categories as seen in algorithms 14 and 15.

Algorithms 23 to 26 are utilised for the job handling discussions since the auto-scalers have specific functions that must be modelled differently.

In order for job processing to commences, the auto-scalers activate the *InitRequiredFunctions*, *JobHandRelist* derived functions and the *jobhandling modules*. This fosters the provision of resources via universes and functions for job initialising. This causes *SystemState* to transition to *busy* as seen in lines 1 to 5.

The auto-scalers exhibited behaviours per their core functions. The behaviours are analysed below.

First, *threshold* and *vmcreate* applied *VmUL* to monitor VM utilization. *VmUL* is utilised differently in the two auto-scalers. In the case of *threshold*, if the current VM utilisation is lower than the average VM threshold, the VMs are destroyed. However, if the current VM utilisation is lower than the average VM threshold but the VM is the last VM being processed; the duration period is extended by one hour to receive a new job before the VM

Algorithm 25 Vmopt Jobs handling

Require: AResource

1: **while** $InitReqFunctions = IRF_{active}$ **do**
2: $JobHandMOD_{multimode} := active$
3: **if** $(JobHandReslist = active) \wedge RVM = Q_{max}$ **then**
4: $ReqResources := true \wedge SystemState(j, p) := busy$
5: **end if**
6: **if** $RVM >= Q_{min}$ **then**
7: $SystemRequest(p, ar) := true \wedge ReqResources := true$
8: $Joblauncher(Jobhandler(j, vm)) := true$
9: $SystemState(j, p) := busy$
10: $SimulationDuration := SD_{max} \wedge AveragUtilPM := AUPM_{max}$
11: $AveragQueTime := AQT_{max}$
12: **end if**
13: **end while**

is destroyed. This is seen in lines 6 to 10 of algorithms 23.

In the case of *Vmcreate*, if the maximum utilisation of VMs is greater than the expected VM threshold, more VMs added are created. Also, if the current threshold is greater than the expected VM threshold, more VMs are created. This causes *SystemState* to transition to *busy* as seen in lines 6 to 11 of algorithms 24.

Second, *Vmopt* monitors the VM count of reusable VMs in the VI. If the number reusable VMs are more than and equal to the minimum number expected, the job processing continues until all the jobs are processed as seen in lines 6 to 11 of algorithms 25.

Third, *Pooling* monitors the VM count in the VM pool during job processing. If the number of VMs are more that the minimum expected, more VMs are created. the job processing continues until all the jobs are processed as seen in lines 6 to 11 of algorithms 26. This causes *SystemState* to transition to *busy*.

These job handling refinements are equivalent to the algorithms of our ground model shown 16 to 17 which are reflective of figure 5.5. Also, the refinement satisfies the evaluation criteria discussed in sub-section 6.2.1. This is seen in the application of derived functions to model job initialising to job handling. Also, state changes are seen when the conditions for our function are met, which are reflective of *guarded updates* of control state ASMs.

Algorithm 26 *Pooling Jobs handling*

Require: AResource

1: **while** $InitReqFunctions = IRF_{active}$ **do**
2: $JobHandMOD_{multimode} := active$
3: **if** $JobHandReslist = JHRL_{active} \wedge Vmpool = Q_{max}$ **then**
4: $ReqResources := true \wedge SystemState(j,p) := busy$
5: **end if**
6: **if** $Vmpool >= Min_{\mathrm{Q}}$ **then**
7: $AddVM(j,vm) := true \wedge ReqResources := true$
8: $Joblauncher(Jobhandler(j,vm)) := true$
9: $SystemState(j,p) := busy$
10: $SimulationDuration := SD_{max}$
11: $AveragQueTime := AQT_{max} \wedge AveragUtilPM := AUPM_{max}$
12: **end if**
13: **end while**

6.2.2.5 Rule 5, Job Termination:

Job Termination was evaluated per subsection 5.4.1.5. The *pooling* auto-scaler was utilised to evaluate the job termination phase. The evaluation of job termination required a modelling that showed an interaction of the previously discussed phases.

Initially, *InitRequiredFunctions*, *jobhandling modules* are activated for the provision of *Aresources* via universes and functions towards job initialising and job handling. The *JobHandRelist* is triggered to foster the job handling process. *SystemState* transitions to *busy* as seen in lines 1 to 4 of algorithm 27.

Moreover, the VM count is monitored in the VM pool. If the number of VMs in the VM pool is within the minimum range, more VM are provisioned. This causes the *systemState* to transition to *busy* as seen in lines 6 to 17. The output functions *SimulationDuration* transitions to *SDmax*, *AverageQueTime* to *AQTmax* and *AverageUtilPM* transitions to *AUPMmax* as seen in lines 8 to 10.

Furthermore, while jobs are being processed, a terminate event causes *systemState* transitions to *stopped* and *event* to *terminate* as seen in lines11 to 12. Also, when the jobs generated are exhausted while *SystemRequest* is activated; *event* transitions to *terminate* and *systemstate* to *done* as seen in lines13 to 17.

This *pooling* auto-scaler refinement of job termination is equivalent to the rule 5 of our model shown in figure 5.5, and algorithm 18 to 19. Also, the refinement satisfies the evaluation criteria discussed in sub-section 6.2.1.

Algorithm 27 *Pooling Jobs Termination*

Require: AResource

while $InitReqFunctions = IRF_{active}$ **do**
  $JobHandMOD_{multimode} := active$
  if $JobHandReslist = JHRL_{active} \wedge Vmpool = Q_{max}$ **then**
    $ReqResources := true \wedge SystemState(j,p) := busy$
  end if
  if $Vmpool >= Min_{\mathrm{Q}}$ **then**
    $AddVM(j,vm) := true \wedge ReqResources := true$
    $SystemState(j,p) := busy$
    $SimulationDuration := SD_{max}$
    $AveragQueTime := AQT_{max} \wedge AveragUtilPM := AUPM_{max}$
  end if
  if $SystemState(j,p) = busy \wedge event(task(p)) = terminate$ **then**
    $SystemState(j,p) := stopped$
  else if $SystemRequest(p,ar) \wedge \nexists j \in JOB \wedge \nexists p \in PROCESS$ **then**
    $Event(t) := Terminate$
    $SystemState(j,p) := done$
  end if
end while

This is seen in the application of derived functions to model job initialising to job termination. State changes are seen when the conditions for function are met, which are reflective of *guarded updates* of control state ASMs. Also, there is interaction between the phases of our model via the application of universes and functions.

In conclusion, this refinement enabled us to assess the applicability of our model to the auto-scalers offered with DISSECT-CF. Also, it enabled us to evaluated their VM provision behaviours. Now, we move ahead to apply our model to other auto-scalers.

6.3 Adoption of Astam with other auto-scaling algorithms

The literature in section 2.7 analysed past auto-scalers mechanisms, and selected [174] for in-depth analysis with our *Astam* model. The algorithms of Yang et al.'s work have been made public; hence it was possible to apply our model. This auto-scaler presents a typical auto-scaling approach using workload prediction, as well as horizontal and vertical scaling. Therefore, it

Algorithm 28 *LoadPredict Jobs Initializing*

Require: AResource

1: **if** $InitReqFunctions = IRF_{active} \wedge SystemRequest(p, ar) = true$ **then**
2: $WorkloadPrediction := PWL_{active} \wedge ReqResources := true$
3: $SystemState(j, p) := active$
4: **end if**
5: **while** $\exists s \in SERVICE \wedge servRequest(s, ar) \wedge SystemRequest(p, ar)$ **do**
6: **for all** $s \in SERVICE$ **do**
7: $ReqResources := true \wedge Joblauncher(Jobhandler(j, vm)) := true$
8: $Vmpool_t := Q_{min} \wedge VmUL_t := VMut_{min} \wedge RtScaling_i := active$
9: $Pre\text{-}scaling_{(t+1)} := active \wedge NumofSerReq_t := Num_{min}$
10: **end for**
11: $SystemState(j, p) := active$
12: **end while**

was possible to analyse and classified it as a *multimode* auto-scaler due to its specific features. Also, our model's usability was evaluated by adopting this auto-scaling mechanism.

The discussions here do not include the initial phase and job termination because they are the same for all auto-scalers. Let us begin our discussion with job initialising.

Job Initialising The job initialising phase is evaluated per Rule 2 of our *Astam* model discussed in sub-section 5.4.1.2. The *InitRequiredFunctions* function is applied to foster *Aresource* provisions via the interactions of universes and function. Workloads are predicted during job initialising which causes the *systemstate* transition to *active* as seen in lines 1 to 3 of algorithm 28 and in figure 6.2. *SystemRequests* and *ReqResources*are activated which causes jobs to be mapped to VMs and installed as tasks.

Services are provisioned for users as the number of service requests increase. This causes the provision of more VMs in the VM pool. Resource scaling and VM utilisation ($VmUL$) are activated as seen in lines 5 to 8.

Also, pre-scaling $(t + 1)^{th}$ Interval is activated as the provision of user services increases which causes *systemstate* to transition to *active* as seen in lines 9 to 11.

This refinement is equivalent to the job initialising phase of our ground model's algorithm 9. Since *systemstate* transitions to *active* as seen in

Algorithm 29 *LoadPredict Jobs Queuing*

Require: AResource

1: **if** $InitReqFunctions = IRF_{active} \wedge SystemRequest(p, ar) = true$ **then**
2: $ReqResources := true \wedge SystemState(j, p) := active$
3: $WorkloadPrediction := PWL_{active} \wedge ReqResources := true$
4: **end if**
5: **if** $QueReslist = QRL_{active}$ **then**
6: $SystemState(j, p) := active$
7: **end if**
8: **while** $\exists s \in SERVICE \wedge servRequest(s, ar) \wedge SystemRequest(p, ar)$ **do**
9: **for all** $s \in SERVICE$ **do**
10: **if** $VMCount \leqslant Num_{min}$ **then**
11: $ReqResources := false$
12: $Joblauncher(Jobhandler(j, vm)) := false$
13: $Vmpool := Q_{avg} \wedge VmUL_t := VMut_{avg}$
14: $NumofSerReq := Q_{avg}$
15: $SystemState(j, p) := waiting$
16: **end if**
17: **end for**
18: **end while**

figure 5.5. Also, from figure 6.2, we can see that the algorithms for job initialising can be refined back to the ground model in an ASM run.

Job Queuing This phase is evaluated per rule 3 of our *Astam* model discussed in sub-section 5.4.1.3. *InitRequiredFunctions* function is applied to foster *Aresource* provisions via the interactions of universes and function. The prediction of workloads is activated which causes the *systemstate* to transition to *active* as seen in lines 1 to 3 of algorithm 29. Also, *QueReslist* is activated to monitor the functions required for job queuing.

Services are provision per the requests of *SystemRequests*, and *ReqResources* causes jobs to be mapped to VMs and installed as tasks. The number of VMs provisioned is monitored via *VMcount*. When the threshold of VM count reduces, the *Joblauncher* and *ReqResources* are updated to *false*. Which signifies VMs shortage as seen in lines 8 to 15. The states of the VM pool, VM Utilisation and the number of service request were updated to average utilisation and quantities. The *systemstate* transitioned to *waiting* as seen in line 16.

The refinement of this phase is equivalent to the job queuing phase

Algorithm 30 *LoadPredict Jobhandling Module* ($JobHandMod_L$)

Require: AResource

1: **if** $InitReqFunctions = IRF_{active} \wedge SystemRequest(p, ar) = true$ **then**
2: $SystemState(j, p) := active$
3: **end if**
4: **if** $QueReslist = QRL_{active}$ **then**
5: $WorkloadPrediction := PWL_{active} \wedge ReqResources := true$
6: $SystemState(j, p) := active$
7: **end if**
8: **while** $\exists s \in SERVICE : servRequest(s, ar) = true$ **do**
9: **for all** $s \in SERVICE = true$ **do**
10: **if** $VMCount \leqslant Num_{min}$ **then**
11: $ReqResources := false$
12: $Joblauncher(Jobhandler(j, vm)) := false$
13: $Vmpool := Q_{avg} \wedge VmUL_t := VMut_{avg}$
14: $NumofSerReq := Q_{avg}$
15: **end if**
16: $SystemState(j, p) := waiting$
17: **end for**
18: **if** $VmRequest(j, vm) = active$ **then**
19: $ReqResources := true$
20: **end if**
21: **end while**

of our ground model, shown in algorithm 12 of our model, reflective of our ground model's figure 5.5. Since the system states transitions to *waiting*.

Job Handling This phase is evaluated per rule 4 of our *Astam* model discussed in sub-section 5.4.1.4. In order to evaluate this auto-scaler, the *jobhandling module* for *multimode* auto-scalers is adopted and applied to optimize VM selection as seen in algorithm 30.

Lines 1 to 16 have been described as part of job queuing in algorithm 29; therefore VM selection is activated via *VmRequest*. This caused VMs and Jobs to be mapped and installed as tasks to continue job processing as seen in lines 18 to 21 of algorithm 30.

The modelling of the auto-scaler by [174] requires the application of *InitReqFunctions* to initiate job processing. This activates the Workload prediction function and *jobhandling module*. This causes *systemstate* to be updated to *active* as seen in lines 1 to 5 of algorithm 31.

Algorithm 31 *LoadPredict Jobs handling*

Require: AResource

1: **if** $InitReqFunctions = IRF_{active} \wedge SystemRequest(p, ar) = true$ **then**
2: $WorkloadPrediction := PWL_{active} \wedge ReqResources := true$
3: $JobHandMod_L := active \wedge Joblauncher(Jobhandler(j, vm)) := true$
4: $SystemState(j, p) := active$
5: **end if**
6: **while** $JobHandReslist = JHRL_{active}$ **do**
7: **if** $VmUL_t > VMut_{max}$ **then**
8: $SystemState(j, p) := busy$
9: **end if**
10: **for all** $vm \in VM$ **do**
11: **if** $VmUL_t > VMut_{max}$ **then**
12: $selfhealing_{SU} := active$
13: **end if**
14: **if** $VmUL_t > VMut_{max}$ **then**
15: $AR_{SU} := active$
16: **end if**
17: **if** $VmUL_t < VMut_{min}$ **then**
18: $AR(VR)_{SD} := active$
19: **end if**
20: **end for**
21: $AveragUtilPM := AQT_{max}$
22: $SimulationDuration := SD_{max} \wedge AveragQueTime := AUT_{max}$
23: $SystemState(j, p) := busy$
24: **end while**

JobHandReslist is activated to foster the provision of resources via universes and functions for job handling which causes the VMs utilisation levels to increase. A maximum utilisation threshold causes *system-state* to transition to *busy* as seen in lines 6 to 9.

Self-healing scaling up is activated to ensure that more *Aresources* are provisioned to support the increased VM utilisation threshold as seen in lines 10 to 13. This occurs when more VMs are provisioned with an accompanying increase in VM utilisation threshold.

Also, the high VM utilisation threshold causes resource-level scaling up to be activated. This utilises unallocated *Aresources* to scale up the VMs as seen in lines 14 to 16.

However, when the VM utilisation threshold decreases, *Aresources* scal-

Algorithm 32 *Virtual Resource scaling down*

Require: AResource

1: **while** $JobHandMod_L = busy$ **do**
2: $SystemState(j, p) := busy$
3: **while** $JobHandReslist = JHRL_{active}$ **do**
4: **if** $VmUL_t < VMut_{max}$ **then**
5: $VMLevel_{SD} := active$
6: **end if**
7: **if** $VmUL_t < VMut_{min}$ **then**
8: $RLevel_{SD} := active$
9: **end if**
10: $SystemState(j, p) := busy$
11: **end while**
12: **end while**

ing down is activated as seen in lines 17 to 19. This causes the outputs of job processing to be generated. The simulation duration average queuing time and average utilisation of PMs are is updated to *maximum* values. The *systemstate* is updated to *busy*.

This refinement is equivalent to the job handling phase of our ground model shown in figure 5.5.

The auto-scaler by [174] employs three specialised scaling operations in this phase.

Virtual Resource Scaling Down: During job handling, when the VM utilisation threshold is below the expected maximum threshold, VM level scaling down is activated. This causes unused *Aresources* of VMs to be scaled down as seen in lines 3 to 6 of algorithm 32. Also, when the state of the VM utilisation threshold still remains the same, the resource level scaling down is activated as seen in lines 7 to 9. *SystemState* remained updated to *busy* throughout this operation.

Pre-scaling at $(t+1)^{th}$ interval: Moreover, the number of service requests are predicted during job handling. This activated the computation of VM utilisation threshold at $(t+1)^{th}$ intervals as seen in lines 3 to 5 of algorithm 33. When the VM utilisation threshold at $(t+1)^{th}$ interval is greater than the maximum VM utilisation threshold, cost aware scaling up is activated as seen in

Algorithm 33 *Pre–scaling at the* $(t+1)th$ *interval*

Require: AResource

```
1: while JobHandMod_L = busy do
2:    SystemState(j, p) := busy
3:    while JobHandReslist = JHRL_active do
4:       Predict − NumofSerReq_{t+1} := active
5:       Calculate − VmUL_{t+1} := active
6:       if VmUL_{t+1} > VMut_max then
7:          Cost − Aware_{P−SU} := active
8:       end if
9:       SystemState(j, p) := busy
10:   end while
11: end while
```

Algorithm 34 *Cost–aware Pre–scaling up*

Require: AResource

```
1: while JobHandMod_L = busy do
2:    SystemState(j, p) := busy
3:    while JobHandReslist = JHRL_active do
4:       VmLevel_SU := active
5:       if NumOfRequests > 0 then
6:          if Aresource < NumOfRequests then
7:             SmallestVM_SU := active
8:          else
9:             Rlevel_SU := active ∨ VMlevel_SU := active
10:         end if
11:      end if
12:      SystemState(j, p) := busy
13:   end while
14: end while
```

lines 6 to 8. *SystemState* remained updated to *busy* throughout this operation.

Cost-aware Pre-scaling up: Moreover, during VM level scaling up, when the number of user requests is greater than zero and the *Aresources* provisioned are not sufficient to handle user requests. The VM with the smallest capacity is activated as seen in lines 3 to 7 of algorithm 24. Conversely, if the *Aresources* are sufficient, a comparison between resource level scaling up and VM level scaling up

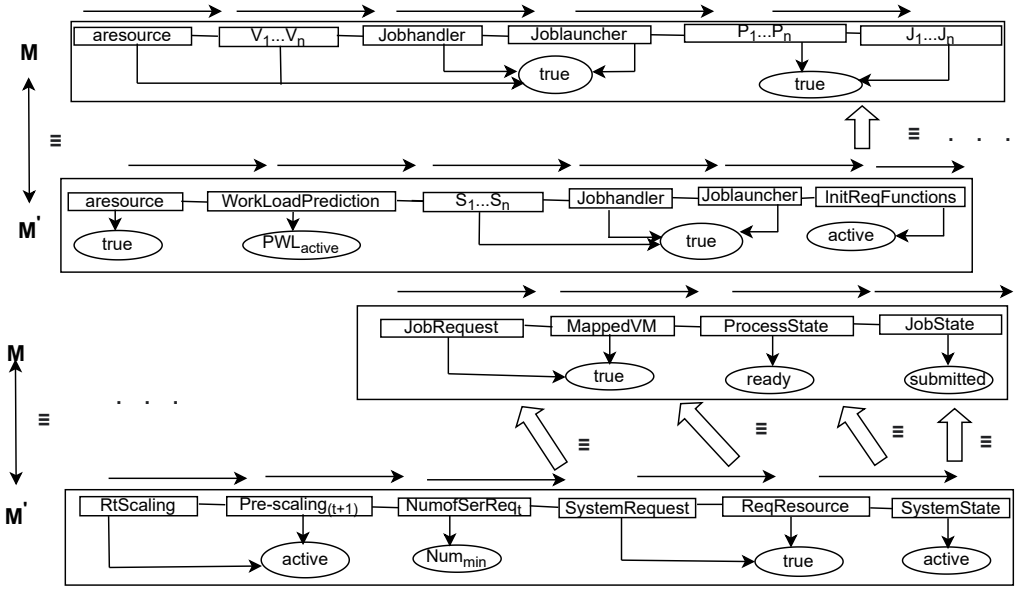


Figure 6.2: Workload Predicted Auto-Scaler Job Initialising

is made to select the appropriate option as seen in lines 8 to 10.The *SystemState* remained updated to *busy* throughout this operation.

This refinements satisfies our evaluation criteria since derived functions were applied to evaluate job initialising to job handling of [174]. Also, the *systemstate* transitioned during the modelling of the phases. The refinements of the specialised operations are equivalent to our ground model.

Now, Let us discuss the verification of *Astam* with CTL Properties.

6.4 Verification of Astam with CTL Properties

The verification of *Astam* (which is part of step 4 of our model's design) was achieved per definition 6 of transition systems and our verification notations described in sub-section 5.3.3. This is accomplished to ensure that our ASM refinements are equivalent to our ground model. Equations 6.1 and 6.2 show how the CTL properties were applied to verify key *universes* and *functions* that are reflective of our ground model in figure 5.5. Equations 6.1 represents the state transitions in our ground model while figure 6.2 represent the state transitions in the refinements.

Now, let us discuss the equations beginning with equation 6.1 .

$$
\begin{aligned}
&1.\, Init.P : M, s_i \vDash jhf \wedge prf \wedge jrf \wedge mvf \wedge mjf \\
&2.\, JobI. : M, s_i \vDash jhf \wedge prf \wedge jrf \wedge mvf \wedge mjf \\
&\quad \rightarrow AX(jht \wedge prt \wedge jrt \wedge mvt \wedge mjt) \\
&3.\, JobQ. : M, s_i \vDash EF(jht \wedge prt \wedge jrt \wedge mvt \wedge mjt) \\
&\quad \rightarrow EX(jhf \wedge prf \wedge jrf \wedge mvf \wedge mjf) \\
&4.\, JobH. : M, s_i \vDash AG(jhf \wedge prf \wedge jrf \wedge mvf \wedge mjf) \\
&\quad \rightarrow AX(jht \wedge prt \wedge jrt \wedge mvt \wedge mjt) \\
&5.\, JobT. : M, s_i \vDash EF(jht \wedge prt \wedge jrt \wedge mvt \wedge mjt) \\
&\quad \rightarrow EX(jht \wedge prt \wedge jrt \wedge mvt \wedge mjt)
\end{aligned}
\tag{6.1}
$$

According to equation 6.1, in line 1 (which represents the Initial phase of our model) the universes and functions are updated to *false*. This is reflective of the inactivity at the initial phase where the *systemState* transitions to *idle* as seen in algorithms 5 and 6.

In line 2, (Job initialising) the functions which were initially at a *false* state, along all paths in the next state transition to *true* where *systemState* was updated to *active*. This is reflective of algorithms 8 and 9.

In line 3 (Job queuing), there exists a path in the future where functions currently updated to *true* transitioning to *false* due to the possibility of VMs depletions. This causes *systemState* to transition to *waiting* which is reflective of algorithms 11 and 12.

In line 4 (Job handling), along all paths and in all future states, once *Aresources* are provisioned and VMs are mapped to jobs, the functions transition from *false* to *true* in the next state (after VM Selection), causing *systemState* transition to *busy*. Which is reflective of algorithm 16.

In line 5 (Job termination), task processing gets terminate when there is an abrupt system interrupt or when all the jobs uploaded are exhausted. Therefore, there exists a possibility of the functions to transition from *true* to *true* in the next future states, fostering job processing to terminate. This causes *systemState* to transition to either *stopped* or *done* as seen in algo-

rithm 18.

$$
\begin{aligned}
&1.\, Init.P : M, s_i \vDash jhf \wedge jlf \wedge srf \wedge rrf \\
&2.\, JobI. : M, s_i \vDash jhf \wedge jlf \wedge srf \wedge rrf \rightarrow AX(jht \wedge jlt \wedge srt \wedge rrt) \\
&3.\, JobQ. : M, s_i \vDash EF(jht \wedge jlt \wedge srt \wedge rrt) \rightarrow EX(jhf \wedge jlf \wedge srf \wedge rrf) \\
&4.\, JobH. : M, s_i \vDash AG(jhf \wedge jlf \wedge srf \wedge rrf) \rightarrow AX(jht \wedge jlt \wedge srt \wedge rrt) \\
&5.\, JobT. : M, s_i \vDash EF(jht \wedge jlt \wedge srt \wedge rrt) \rightarrow EX(jht \wedge jlt \wedge srt \wedge rrt)
\end{aligned}
\tag{6.2}
$$

In line 1 of equation 6.2, the refined functions transition to *false* after the ASM run which caused *systemState* to transition to *idle* which is reflective of the refined initial phase algorithm 7.

In line 2, the functions previously updated to *false* (in line 1), along all paths transition to *true* in the next phase (i.e., job initialising). This caused *systemstate* to transition to *active* as seen in algorithm 10.

In line 3, the functions previously updated to *true* (in line 2), along some paths in the next state due to the possibility of VM shortage, transition to *false*. *Systemstate* is updated to *waiting*. This is reflective of algorithm 13.

In line 4, along all paths (globally), once *Aresources* are provisioned, the functions previously updated to *false* (in line 3), transition to *true* in the next state, when all conditions are right (after VM Selection). *systemState* to transition to *busy* which is reflective of algorithm 17.

In line 5, task processing get terminated in the next possible path for functions previously updated to *true* (in line 4). This happens when the conditions of job termination are met. This causes *systemState* to be updated to either *stopped* or *done*.

The verification process showed, that refinements are equivalent to our ground model as the functions and state transitions exhibited similar VMs provision behaviours. Additionally, this equivalence of state transitions aligns with definitions 5 to 4 of *Schellhorn's* theorem and *Börger's refinement* of ASM verification.

Now let us move on to discuss the validation of *Astam* with CoreASM.

6.5 Validation of Astam with CoreASM

6.5.1 Introduction

Our ASM model's validation (step 5 of our model's design) was achieved through the creation of test cases from our ground model algorithms and

their refinements. These test case were developed on CoreASM (plug-in available for the Eclipse IDE).

The processes utilized for validating the test cases, the expected results and the subsequent results are described below. The test cases for validating *Astam* (i.e., *coreasm specifications*) were grouped into three major derived functions (also know as modules). They are: job initialising, queuing, and handling specifications.

The test cases were designed as interactive sequences with suitable assessment criteria to describe the expectations of our model's states. This was accomplished to see if specified assertions hold in given states. The test cases were processed and examined if all the assertions were satisfied. A satisfied assertion finished with a *pass* verdict. However, as soon as an assertion was not satisfied, the simulation was interrupted, reporting a violation.

At each step, the simulator (CoreASM) performed update monitoring to ensure that all states were updated. Our validation goals are:

- To assess the interactions of the phases of our ground model via the application of universes and signatures.
- To examine the application of derived functions (modules) as refinements of our ground model.
- To assess the application of *guarded updates* (which are reflective of control state ASMs) to ensure equivalence (between ground model algorithms and their refinements).

Let us now outline the steps employed to examine the application of our validation goals on the test cases.

- Assess the test cases to ascertain the the provision of resources via the application of *universes* and *functions* for all the job execution phases.
- Monitor VM counts to ascertain the queuing of job for VMs (at step 2), and VM selection when jobs are mapped to VMs.
- Report all state changes during this process. The expected state update should be *idle* for the initial phase, *submitted* or *active* for job initialising, *waiting* for job queuing, *busy* for job handling and either *stopped* or *done* for job termination.

We applied the validation goals on the test cases developed for the refined algorithms of our ground model's algorithm to proved their equivalence. Table 6.1 depicts the relationship between our algorithms and the validated CoreASM Specifications.

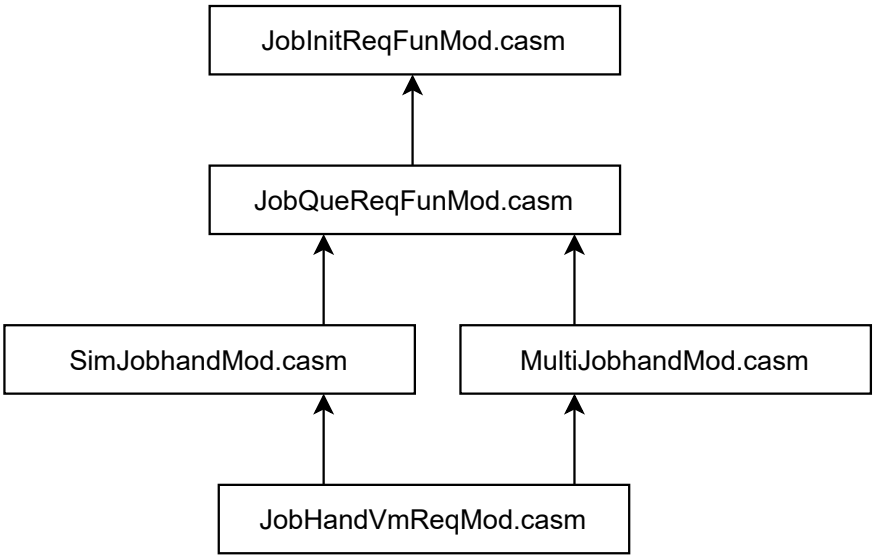


Figure 6.3: ASM Validation Modules

6.6 Creation of Astam Modules for Model Validation

CoreASM modules (CoreModules) were created to ensure modularisation amongst the CoreASM specification. Also, they were developed to support the ASM specifications with *signatures*, *universes*, *functions* required for validation process. The CoreModules were designed with ASM application rules which aligned with the control state ASM definition 2 to ensure that, once the conditions for state transitions are met, system states are updated accordingly. The conditions in the test cases are designed to be set to *true*. If this condition is met, system states are automatically updated.

Three key modules were applied to support our model validation as seen in figure 6.3. Figure 6.3 represents the actual file names of the modules with the ASM specification file extensions (.casm). The arrows represents the levels of dependency of each file from the bottom to the top. The topmost file is the module for job initialising, which is followed by the job queuing module. The *jobhandler module* for *SimpleMultimode* auto-scalers are on the same level, which are followed by a general *jobhandler module* with VM selection integrated refinements. The *MultiJobHandMod* and *SimJobHandMod* modules were validated and applied to the *Multimodes* and *Simple* auto-scalers as seen in MultiJobHandMod.casm and SimJobHandMod.casm. The two modules were validated separately.

In order to validate the modules, the test cases were scrutized for the provision of *aresources* and job queuing resources via the application of the *InitReqFunctions*, *VMcount* and *QueReslist* functions. All the test cases generated positive assertion verdicts with the expected state transtions. The CoreModules were then applied in the validation of the other test cases representing the phases of our ground model. Let us discuss the ASM validation rules in the next section.

6.7 Astam Transition Rules Validation with CoreModules

In this section, the validation of the ASM rules is discussed extensively to highlight the applicability of our model. The *Simple* and *Multimode* auto-scalers were validated separately. Also, the ground model and their refinements were all validated separately. As mentioned above, the test cases of the refinements are expected to show equivalence to the test cases of our ground model. Also, the results for the ground model and the refinement are expected to show positive assertion verdicts. Details of the validation of *Astam's* transition rules are described below.

6.7.1 Initial Phase

This phase was validated following rule 5.4.1.1 of our model as seen in the *JobInitFunMod.casm* file. A series of test case assessements were done to validate the initial phase of our model. The ground model and its refinement were checked for the provision of *universes* (including specific auto-scalers universes) and *functions*. The *jobRequest* and processRequest were utilised for the ground model; while *InitReslist*, *SystemRequest* and *ReqResource* functions were employed for refinement. The *jobState*, *jobTime* and *systemstate* were updated to *idle*.

6.7.2 Job Initialising

This phase was validated following rule 5.4.1.2 of our model. The validation required a re-run of the initial phase to ensure coherence. This fostered the mapping of jobs to VMs, and their installation as tasks. The test cases were assessed for the provision of resources via universes and functions. The *systemstate* transitions to *active*; and *jobState* was updated to *submitted*.

Table 6.1: Algorithms with Associated ASM Specification Files (AN: Algorithm Numbers)

AN	Algorithms	File Names
5	Simple Initial Phase	SimpleInitialPhase.casm
6	Multimode Job Initial	MultimodeInitialPhase.casm
7	Refined Initial Phase	RefinedInitialPhase.casm
8	Simple Job Initialising	SimpleJobJobInitialising.casm
9	Multimode Job Initialising	MultimodeInitialising.casm
10	Refined Job Initialising	RefinedJobInitialising.casm
11	Simple Job Queuing	SimpleJobQue.casm
12	Multimode Job Queuing	MultimodeJobQue.casm
13	Refined Job Queuing	RefinedJobQue.casm
14	Simple Jobhand.	SimJobHandMod.casm
15	Multi. Jobhand. Mod.	MultiJobHandMod.casm
16	Job Handling	JobHandling.casm
17	Refined Job Handling	RefinedJobHandling.casm
18	Job Termination	JobTermination.casm
19	Refined Job Termination	RefinedJobTermination.casm
20	LoadPred. Jobs Init	LoadPredJobInit.casm
21	LoadPred. Jobs Que	LoadPredJobQue.casm
22	LoadPred. Jobh. Mod	LoadPredJobHandMod.casm
23	LoadPred. Job hand.	LoadPredJobHand.casm
24	Virt. Res. Scal. down	LoadVirtResScalDown.casm
25	Pre-scaling Int.	LoadPreScalInterval.casm
26	Cost-Aware Pre-scal.	LoadCostAwarePSU.casm
27	Threshold Initial Phase	ThresholdInitPhase.casm
28	Vmopt Jobs Init.	VmoptJobInit.casm
29	Threshold Jobs Que.	ThresJobQue.casm
30	Thres. Jobhanding	ThresJobHand.casm
31	VmCreate Jobs handling	VmCreateJobHand.casm
32	Vmopt Jobs handling	VmoptJobHand.casm
33	Pooling Jobs handling Mod	PoolJobHandling.casm
34	Pooling Jobs Term Mod	PoolJobTermination.casm

6.7.3 Job Queuing

The test cases created to validate job queuing applied rule 3 discussed in sub-section 5.4.1.3. The test cases were assessed for the provision of resources via universes and functions. The *VMCount* function was applied to monitor the number of VMs during job processing. The *systemstate* and *JobState* were updated to *waiting*.

6.7.4 Job Handling

The test case created to validate Job handling applied rule 4 of our model discussed in sub-section 5.4.1.4. The validation applied the *MultiJobHandMod* and *SimJobHandMod* modules described in section 6.6. Derived functions (*InitReqFunctions* in the *JobInitFunMod.casm* files) were utilised to provision *universes* and *functions* from the previous phases. Also, job queuing and VM selection were monitored via the *jobhandling modules* in *JobHandVmReqMod.casm* file. In the case of the refinement, the *Jobhandling module* mentioned above was applied.

6.7.5 Job Termination

Job termination was validated following sub-section 5.4.1.5 of our model. The test cases developed for this phase checked the conditions for job termination. Previous phases were re-executed and validated sequentially to lead to job termination. The primary conditions for job termination were validated. The test case (with universes designed to stop working midway) was assessed for system interruptions. Also, the test case (designed to monitor the availability of jobs) was monitored for the quantity of jobs available at the end of task processing. The *jobtime* was updated to *completed*. Also, *systemstate* transitioned to *done* (as expected).

In conclusion, the test cases developed from our ground model algorithms and their refinements aligned with the validation goals discussed in section 6.5.1. In general, the test cases of the refinements showed equivalence to the test cases of our ground model. Also, the results for the ground model and the refinement showed positive assertion verdicts. Now let us summarize this chapter.

6.8 Summary

In this chapter, we discussed the processes utilised to evaluate, verify and validate our *Astam* Model. The evaluation section applied our *Transition Rules*

and evaluation criteria to examine the algorithms developed from available auto-scalers. The evaluation extended to other auto-scalers whose algorithms were made public. The evaluation showed the equivalence of the refinements (of the auto-scalers) to our *ground model* and their alignment with the evaluation criteria. For the model verification, the Computational Tree Logic (CTL) formulae were applied to our notations to ensure the correctness of our refinements. Moreover, we developed test cases from the formalized algorithms of the auto-scalers and validated on the CoreASM simulator. The test cases were grouped into several specifications and validated.

Finally, the discussions of this chapter provided sufficient information to show that our *Astam* model was designed to reflect the observations of the investigations carried out in section 5.2. This shows that, although the auto-scalers were designed on different frameworks, they exhibit similar VM provision behaviours during job processing. These behaviours are not only limited to cloud computing framework but also the MR programming model which have been proved via research to support auto-scaling mechanisms.

To allow for further scrutiny and reuse, the validated CoreASM Specifications are available in the Auto-Scaling-ASM repository on Github[1]. Now let us move on to conclude our dissertation.

[1] `https://github.com/EbenezerKomlaGavua/Auto-Scaling-ASM`

Chapter 7
Conclusion

This dissertation focused on the application of cloud computing simulators and the modelling of resource provisions to meet user demands. As such, three research efforts were carried out. These are discussed below.

The initial research effort focused on the evaluation and comparisons of MR simulators. Literature was reviewed systematically to identify simulator features that were specific to cloud computing in general. These cloud-oriented features were again revised to MR specific features. These were utilised as evaluation criteria (classification framework) to analyse the strengths and weaknesses of MapReduce related simulators. The classification framework showed that MR simulators support features, including data-intensive applications, application configuration management, and suitable infrastructural implementation. The research showed that the older simulators developed are not suitable for current research. This was due to the fact that they lack most of the features required to carry out MR research. However, they provided the platform for simulators developed recently and do support more features for efficient research. The investigation enabled the recommendations to be provided to users concerning the choice of MR simulators for research and development. Therefore, the results of the initial research effort simplifies the selection of MapReduce simulators for researchers. The research fostered the adherence to systematic literature review procedures. Since initially, several simulators were considered (including peer-to-peer simulators) which were not MR specific. These were removed due to the absence of the expected features. This prolonged the investigation process which will not be repeated in future research activities. Also, with the further improvements on several MR simulators. The selection of simulators will be much simpler with this classification framework.

As the strengths and weaknesses of the simulators were been analysed, one challenge with the MR Hadoop became apparent (i.e.,speculative execution). Since some simulators could simulate speculative execution while

others could not. Therefore, this challenge was further investigated, which led to the design of an approach to tackle speculative execution in MapReduce Hadoop. Three algorithms were developed towards job performance improvement. The algorithms are: (*i*) snapshot capturing (*ii*) task performance monitoring and (*iii*) task instance monitoring algorithms. K-means clustering technique was applied to classify captured task run times into two categories (straggler and non-straggler tasks). Also, Silhouette score was implemented on the clustering algorithm as a decision-making tool. This was because K-means tried to classify all data set regardless of their distribution. Thus, the clustering results required validation to determine their goodness of fit. Hence, a silhouette score was applied to determine when to process backup tasks on available nodes. This approach was evaluated on several data centre configurations and against baseline methods. These configurations were selected based on a survey of industry requirements for Hadoop clusters and applications. Experiments were carried out which were directed by three objectives; namely (*i*) the determination of overheads caused by implementing the approach and (*ii*) the job performance improvements (*iii*) comparison with baseline methods. The experiments were carried out on the HDMSG-EXTENSION simulator (one of the simulators evaluated by the first research effort). The experiments showed that (*i*) the overheads caused by applying the proposed approach were reduced faster with large data centres than with smaller data centres. (*ii*) mapper tasks with typically longer task run times had better chances for improvements. (*iii*) Our approach showed improved performance over Hadoop Naive, Longest Approximate Time to End (LATE) and the Self-Adaptive MR Scheduling Algorithms (SAMR) methods. Therefore, adequate Hadoop data centre recommendations have been provided for users who will apply this approach. The design of this approach showed that, it is vital to conduct surveys from industry to ensure that solutions provisioned are applicable to real life and industry.

Furthermore, it was discovered through literature that resource provision was pivotal to the meeting user demands during job processing. However, the issue of evaluating auto-scaling mechanisms to determine existing similar behaviours remained a challenge. Especially when the auto-scalers were developed on different infrastructures. Therefore, an ASM model was develop to analyse resource provision behaviours on cloud mechanisms.

Initially, investigations about the VM provision behaviours of the auto-scaling mechanisms offered with the DISSECT-CF simulator was accomplished as a motivation towards the model. Two categories of auto-scalers (*simple* and *multimode*) and their integral components were identified. These components were required in the design an auto-scaler and could be modelled with ASMs. Then, ASMs were applied to model and analyse the VM

provision behaviours of auto-scaling mechanism. To achieve this, algorithms were generated from the auto-scalers investigated earlier, according to the ASM model refinement method. These algorithms were formalized to reflect job execution phases, which represented our ground model's Transition Rules. The initial algorithms were refined according to the ASM refinement method, and compared to their refinements according to *Börger's refinement* to check for equivalence. The model was utilised to evaluate other auto-scalers to ensure applicability. The CTL formulae was applied on the model's notations to check for correctness. CTL was utilised because it is generally efficient in model checking. The verification of the model shows that it is readily usable for modelling other solutions. Test cases were developed and validated on the CoreASM (plug-in available for the Eclipse IDE) from the formalised algorithms according to validation rules. The validation rules were created in accordance with the ground model's Transition Rules and ASM *guarded* updates. The application of ASM technique has foster a rigorous analysis of the auto-scalers' algorithms which confirms the initial investigations carried out. This shows that if such techniques are applied to most experiments in industry, concrete reasons can be provided for the reliability of products.

Although this research has achieved our research aims, a couple of tasks are left to be accomplished. Let us now discuss our future research direction.

7.1 Future Research Direction

As future work, we plan to design an auto-scaling algorithm which will be implemented on MapReduce Hadoop. Our Snapshot capturing algorithm will be applied to foster a comparison with the job performance approach. Also, a couple of classification and clustering techniques will be applied to provide further extensions.

Furthermore, we plan to implement CTL properties on test cases as an extension to our ASM verifications. This study will be achieved via available model-based testing and verification approaches. Moreover, developing an ASM unified behavioural model for auto-scaled IoT applications will be considered. This will aim at fostering the assessment of the interaction of IoT applications during their service delivery in clouds. Also, we will consider the applicability areas for our ASM model through more job-related scenarios. These experiments will ensure the modelling of the behaviour of jobs.

Our research efforts in our dissertation has fostered the development of three major contributions to science. Now, let us outline these contributions.

7.2 Contributions to Science

This work contributes to the field of Cloud Computing, Distributed Systems and Software Engineering.

Thesis I: *MaReClass allows the evaluation of simulators by identifying MapReduce specific criteria required to render detailed simulations complete.* MaReClass simplifies the selection of MapReduce simulators with features relevant to the choice of research. The framework fosters systematic analysis of the strengths and weaknesses of mapreduce simulators. [P4, P5]

Thesis II: *Haspeck can improve job performance on MapReduce Hadoop even with the challenges presented by speculative execution. Haspeck* implements snapshots capturing to determine task run times, which inures the early detection and selection of straggler tasks as backup tasks. The application of kmeans clustering with silhouette coefficients fosters the runtime reduction via small overheads for long running mappers and reducers in my solution. [P2, P3, P6]

Thesis III: *Astam is capable of analysing the virtual machine provision behaviours of auto-scaling mechanisms.* The *Astam* model allows the formalisation and comparisons of auto-scaling algorithms from multiple sources. The model can assess formalized auto-scaled algorithms emanating from distinctive architectures to show that they exhibit similar VM provision behaviours. The flexibility of *Astam's* transition rules allows the adoption of auto-scaling mechanisms with extra features besides vertical and horizontal scaling to foster their evaluation. [P1, P7]

7.2.1 Author's Publications During Research

(P1) Ebenezer Komla Gavua, Gabor Kecskemeti: "Formalizing cloud auto-scaling algorithms with the abstract sate machine model" In: Vadászné, Bognár Gabriella; Piller, Imre (eds.) Doktoranduszok Fóruma : Miskolc, 2019. november 21. : Gépészmérnöki és Informatikai Kar Szekciókiadványa Miskolc, Hungary : Miskolci Egyetem Tudományos és Nemzetközi Rektorhelyettesi Titkárság (2020) 188 p. pp. 37-43., 7 p.

(P2) Ebenezer Komla Gavua, Gabor Kecskemeti: "Improving MapReduce Speculative Executions with Global Snapshots" In: The 12th Conference of PhD Students in Computer Science: Volume of short papers Szeged, Hungary : Szegedi Tudományegyetem (2020) pp. 62-65. , 4 p. Scientific

(P3) Ebenezer Komla Gavua, Gabor Kecskemeti: "Application of Kmeans and Hierarchical Agglomerative Clustering Techniques on MapReduce" In: Barna, Boglárka Johanna; Kovács, Petra; Molnár, Dóra; Pató, Viktória Lilla (eds.) XXIII. Tavaszi Szél Konferencia 2020. Absztraktkötet: MI és a tudomány jövője Bp, Hungary: Association of Hungarian PHD and DLA Students (2020) 600 p. pp. 354-354., 1 p. Scientific

(P4) Ebenezer Komla Gavua, Gabor Kecskemeti: "A Comparative Analysis and Evaluation of MapReduce Cloud Computing Simulators" In: Waleed, W. Smari (eds.) 2019 International Conference on High Performance Computing & Simulation (HPCS) Piscataway (NJ), United States of America : IEEE (2019) Paper: 222, 8 p. DOI Scopus index

(P5) Ebenezer Komla Gavua, Gabor Kecskemeti: "Evaluation of MapReduce Simulators Towards the Improvement of DISSECT-CF" In: Németh, Katalin (eds.) Tavaszi Szél 2019 Konferencia. Nemzetközi Multidiszciplináris Konferencia : Absztraktkötet Bp, Hungary: Association of Hungarian PHD and DLA Students (2019) 742 p. pp. 429-429. , 1 p. Scientific

(P6) Ebenezer Komla Gavua and Gabor Kecskemeti, "Improving MapReduce Speculative Executions with Global Snapshots" International Journal of Advanced Computer Science and Applications(IJACSA), 14(1), 2023. Web of Science (WoS), (Q3+ Scopus Index), Impact Factor (1.16), Journal Article

(P7) Ebenezer Komla Gavua, Gabor Kecskemeti: "ASM-based Formal Model for Analysing Cloud Auto-Scaling Mechanisms", Int. Journal. of Computing and Informatic (Informatica). Web of Science (WoS) (Q3+ Scopus Index), 47 (2023) 75–96. https://doi.org/10.31449/inf.v47i6.4622.

Bibliography

[1] Arif Ahmed and Abadhan Saumya Sabyasachi. Cloud computing simulators: A detailed survey and future direction. In *IEEE International Advance Computing Conference (IACC)*, pages 866–872, February 2014.

[2] Rizwan Ahmed and Stewart Robinson. Modelling and simulation in business and industry: insights into the processes and practices of expert modellers. *Journal of the Operational Research Society*, 65(5):660–672, 2014.

[3] Auday Al-Dulaimy, Javid Taheri, Andreas Kassler, M Reza Hoseiny Farahabady, Shuiguang Deng, and Albert Zomaya. Multiscaler: A multi-loop auto-scaling approach for cloud-based applications. *IEEE Transactions on Cloud Computing*, 2020.

[4] Md Imran Alam, Manjusha Pandey, and Siddharth S Rautaray. A comprehensive survey on cloud computing. *International Journal of Information Technology and Computer Science (IJITCS)*, 7(2):68, 2015.

[5] Bader Alouffi, Muhammad Hasnain, Abdullah Alharbi, Wael Alosaimi, Hashem Alyami, and Muhammad Ayaz. A systematic literature review on cloud computing security: Threats and mitigation strategies. *IEEE Access*, 9:57792–57807, 2021.

[6] Khaled Alwasel, Rodrigo N Calheiros, Saurabh Garg, Rajkumar Buyya, Mukaddim Pathan, Dimitrios Georgakopoulos, and Rajiv Ranjan. Bigdatasdnsim: A simulator for analyzing big data applications in software-defined cloud data centers. *Software: Practice and Experience*, 51(5):893–920, 2021.

[7] Khaled Alwasel, Devki Nandan Jha, Eduardo Hernandez, Deepak Puthal, Mutaz Barika, Blesson Varghese, Saurabh Kumar Garg, Philip James, Albert Zomaya, Graham Morgan, et al. Iotsim-sdwan: A

simulation framework for interconnecting distributed datacenters over software-defined wide area network (sd-wan). *Journal of Parallel and Distributed Computing*, 143:17–35, 2020.

[8] Mostafa Ameli, Jean-Patrick Lebacque, and Ludovic Leclercq. Simulation-based dynamic traffic assignment: Meta-heuristic solution methods with parallel computing. *Computer-Aided Civil and Infrastructure Engineering*, 35(10):1047–1062, 2020.

[9] Ganesh Ananthanarayanan, Ali Ghodsi, Scott Shenker, and Ion Stoica. Effective straggler mitigation: Attack of the clones. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 185–198, 2013.

[10] Nick Antonopoulos and Lee Gillam. *Cloud computing*. Springer, 2010.

[11] VP Anuradha and D Sumathi. A survey on resource allocation strategies in cloud computing. In *International Conference on Information Communication and Embedded Systems (ICICES)*, pages 1–7, February 2014.

[12] Paolo Arcaini, Roxana-Maria Holom, and Elvinia Riccobene. Asm-based formal design of an adaptivity component for a cloud system. *Formal Aspects of Computing*, 28(4):567–595, 2016.

[13] Sabeur Aridhi, Laurent d'Orazio, Mondher Maddouri, and Engelbert Mephu Nguifo. Density-based data partitioning strategy to approximate large-scale subgraph mining. *Information Systems*, 48:213–223, 2015.

[14] Farag Azzedin and Mustafa Ghaleb. Internet-of-things and information fusion: Trust perspective survey. *Sensors*, 19(8):1929, 2019.

[15] Deepali Bajaj, Urmil Bharti, Anita Goel, and SC Gupta. Paas providers and their offerings. *International Journal of Scientific & Technology Research*, 9(2):4009–4015, 2020.

[16] Mutaz Barika, Saurabh Garg, Andrew Chan, Rodrigo N Calheiros, and Rajiv Ranjan. Iotsim-stream: Modelling stream graph application in cloud simulation. *Future Generation Computer Systems*, 99:86–105, 2019.

[17] Zhaojuan Bian, Kebing Wang, Zhihong Wang, Gene Munce, Illia Cremer, Wei Zhou, Qian Chen, and Gen Xu. Simulating big data clusters for system planning, evaluation, and optimization. In *43rd International Conference on Parallel Processing (ICPP)*, pages 391–400, November 2014.

[18] Alessandro Bianchi, Luciano Manelli, and Sebastiano Pizzutilo. A distributed abstract state machine for grid systems: A preliminary study. In *Proceedings of the Second International Conference on Parallel, Distributed, Grid And Cloud Computing For Engineering, Civil-Comp Press, Ajaccio, France, Paper*, volume 84, 2011.

[19] Alessandro Bianchi, Luciano Manelli, and Sebastiano Pizzutilo. An asm-based model for grid job management. *Informatica*, 37(3), 2013.

[20] André B Bondi. Characteristics of scalability and their impact on performance. In *Proceedings of the 2nd international workshop on Software and performance*, pages 195–203. ACM, 2000.

[21] Dibya Jyoti Bora, Dr Gupta, and Anil Kumar. A comparative study between fuzzy clustering algorithm and hard clustering algorithm. *arXiv preprint arXiv:1404.6059*, 2014.

[22] Egon Börger. The asm refinement method. *Formal aspects of computing*, 15(2-3):237–257, 2003.

[23] Egon Börger. The asm method for system design and analysis. a tutorial introduction. In *International Workshop on Frontiers of Combining Systems*, pages 264–283. Springer, 2005.

[24] Egon Börger and Uwe Glässer. Modelling and analysis of distributed and reactive systems using evolving algebras. In *University of Aarhus*. Citeseer, 1995.

[25] Khaoula Braiki and Habib Youssef. Resource management in cloud data centers: a survey. In *2019 15th international wireless communications & mobile computing conference (IWCMC)*, pages 1007–1012. IEEE, 2019.

[26] James Byrne, Sergej Svorobej, Konstantinos M Giannoutakis, Dimitrios Tzovaras, Peter J Byrne, Per-Olov Östberg, Anna Gourinovitch, and Theo Lynn. A review of cloud computing simulation platforms and related environments. In *International Conference on Cloud Computing and Services Science*, volume 2, pages 679–691. SciTePress, 2017.

[27] Rodrigo N Calheiros, Rajiv Ranjan, Anton Beloglazov, César AF De Rose, and Rajkumar Buyya. Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms. *Software: Practice and experience*, 41(1):23–50, 2011.

[28] Matteo Camilli, Carlo Bellettini, Lorenzo Capra, and Mattia Monga. Ctl model checking in the cloud using mapreduce. In *2014 16th International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*, pages 333–340. IEEE, 2014.

[29] Henri Casanova, Arnaud Giersch, Arnaud Legrand, Martin Quinson, and Frédéric Suter. Versatile, scalable, and accurate simulation of distributed applications and platforms. *Journal of Parallel and Distributed Computing*, 74(10):2899–2917, 2014.

[30] Henri Casanova, Arnaud Legrand, and Martin Quinson. Simgrid: A generic framework for large-scale distributed experiments. In *Tenth International Conference on Computer Modeling and Simulation (UKSIM)*, pages 126–131, April 2008.

[31] Georgios Chantzialexiou, Andre Luckow, and Shantenu Jha. Pilot-streaming: A stream processing framework for high-performance computing. In *2018 IEEE 14th International Conference on e-Science (e-Science)*, pages 177–188. IEEE, 2018.

[32] Qi Chen, Cheng Liu, and Zhen Xiao. Improving mapreduce performance using smart speculative execution strategy. *IEEE Transactions on Computers*, 63(4):954–967, 2013.

[33] Quan Chen, Daqiang Zhang, Minyi Guo, Qianni Deng, and Song Guo. Samr: A self-adaptive mapreduce scheduling algorithm in heterogeneous environment. In *2010 10th IEEE International Conference on Computer and Information Technology*, pages 2736–2743. IEEE, 2010.

[34] Shenghui Chen, Zhiming Fan, Haiying Shen, and Lu Feng. Performance modeling and verification of load balancing in cloud systems using formal methods. In *2019 IEEE 16th International Conference on Mobile Ad Hoc and Sensor Systems Workshops (MASSW)*, pages 146–151. IEEE, 2019.

[35] Weiwei Chen and Ewa Deelman. Workflowsim: A toolkit for simulating scientific workflows in distributed environments. In *2012 IEEE 8th international conference on E-science*, pages 1–8. IEEE, 2012.

[36] Yanpei Chen, Archana Ganapathi, Rean Griffith, and Randy Katz. The case for evaluating mapreduce performance using workload suites. In *IEEE 19th International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 390–399, July 2011.

[37] Maxim Chernyshev, Zubair Baig, Oladayo Bello, and Sherali Zeadally. Internet of things (IoT): research, simulators, and testbeds. *IEEE Internet of Things Journal*, 5(3):1637–1647, 2018.

[38] Dai-Lun Chiang, Sheng-Kuan Wang, Yu-Ying Wang, Yi-Nan Lin, Tsang-Yen Hsieh, Cheng-Ying Yang, Victor RL Shen, and Hung-Wei Ho. Modeling and analysis of hadoop mapreduce systems for big data using petri nets. *Applied Artificial Intelligence*, 35(1):80–104, 2021.

[39] Chams Eddine Choucha, Mohamed Ramdani, Mohamed Khalgui, and Laïd Kahloul. On decomposing formal verification of ctl-based properties on iaas cloud environment. In *ICSOFT*, pages 544–551, 2020.

[40] Antoine Cornuéjols, Cédric Wemmert, Pierre Gançarski, and Younès Bennani. Collaborative clustering: Why, when, what and how. *Information Fusion*, 39:81–95, 2018.

[41] M David. Sasreduce-an implementation of mapreduce in base/sas. *Whitehound Limited, UK. Online at http://support. sas. com/resources/papers/proceedingsl4/15072014. pdf*, pages 1–16, 2014.

[42] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: a flexible data processing tool. *Communications of the ACM*, 53(1):72–77, 2010.

[43] Elif Dede, Zacharia Fadika, Jessica Hartog, Madhusudhan Govindaraju, Lavanya Ramakrishnan, Dan Gunter, and R Canon. Marissa: Mapreduce implementation for streaming science applications. In *2012 IEEE 8th International Conference on E-Science*, pages 1–8. IEEE, 2012.

[44] Ümit Demirbaga. Iotsim-osmosis: A framework for modelling & simulating iot applications over an edge-cloud continuum. 2020.

[45] Ankita Desai, Rachana Oza, Pratik Sharma, and Bhautik Patel. Hypervisor: A survey on concepts and taxonomy. *International Journal of Innovative Technology and Exploring Engineering*, 2(3):222–225, 2013.

[46] Rajshree A Deshmukh, HN Bharathi, and Amiya K Tripathy. Parallel processing of frequent itemset based on mapreduce programming model. In *2019 5th International Conference On Computing, Communication, Control And Automation (ICCUBEA)*, pages 1–6. IEEE, 2019.

[47] Snehsudha Popatrao Dhage, Tanpure Renuka Subhash, Rutuja Vilas Kotkar, Prachi Dattatray Varpe, and Sonali Sanjay Pardeshi. An overview-google file system (gfs) and hadoop distributed file system (hdfs). *SAMRIDDHI: A Journal of Physical Sciences, Engineering and Technology*, 12(SUP 1):126–128, 2020.

[48] Siddhi Dhamale. Big data a big problem. `https://siddhi-dhamale03.medium.com/big-data-a-big-problem-769cdb475b45`, 2022. [Online; accessed April 23, 2023].

[49] Linlin Ding, Guoren Wang, Junchang Xin, Xiaoyang Wang, Shan Huang, and Rui Zhang. Commapreduce: An improvement of mapreduce with lightweight communication mechanisms. *Data & Knowledge Engineering*, 88:224–247, 2013.

[50] dzone. Hadoop cluster capacity planning of data nodes for batch and in-memory processes. `https://dzone.com/articles/hadoop-cluster-capacity-planning`, 2023. [Online; accessed April 23, 2023].

[51] Jaliya Ekanayake, Hui Li, Bingjing Zhang, Thilina Gunarathne, Seung-Hee Bae, Judy Qiu, and Geoffrey Fox. Twister: a runtime for iterative mapreduce. In *Proceedings of the 19th ACM international symposium on high performance distributed computing*, pages 810–818. ACM, 2010.

[52] Jaliya Ekanayake, Shrideep Pallickara, and Geoffrey Fox. Mapreduce for data intensive scientific analyses. In *IEEE Fourth International Conference on eScience*, pages 277–284, December 2008.

[53] Kalpana Ettikyala and Y Rama Devi. A study on cloud simulation tools. *International Journal of Computer Applications*, 115(14), 2015.

[54] Alexandros Evangelidis, David Parker, and Rami Bahsoon. Performance modelling and verification of cloud-based auto-scaling policies. *Future Generation Computer Systems*, 87:629–638, 2018.

[55] Zacharia Fadika, Elif Dede, Madhusudhan Govindaraju, and Lavanya Ramakrishnan. Mariane: Mapreduce implementation adapted for hpc environments. In *Proceedings of the 2011 IEEE/ACM 12th International Conference on Grid Computing*, pages 82–89. IEEE Computer Society, 2011.

[56] Zacharia Fadika, Elif Dede, Jessica Hartog, and Madhusudhan Govindaraju. Marla: Mapreduce for heterogeneous clusters. In *Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012)*, pages 49–56. IEEE Computer Society, 2012.

[57] Zacharia Fadika and Madhusudhan Govindaraju. Lemo-mr: Low overhead and elastic mapreduce implementation optimized for memory and cpu-intensive applications. In *2010 IEEE Second International Conference on Cloud Computing Technology and Science*, pages 1–8. IEEE, 2010.

[58] Fairouz Fakhfakh, Hatem Hadj Kacem, and Ahmed Hadj Kacem. Simulation tools for cloud computing: A survey and comparative study. In *2017 IEEE/ACIS 16th International Conference on Computer and Information Science (ICIS)*, pages 221–226. IEEE, 2017.

[59] Gilles Fedak, Haiwu He, and Franck Cappello. Bitdew: a programmable environment for large-scale data management and distribution. In *Proceedings of the 2008 ACM/IEEE conference on Supercomputing*, page 45. IEEE Press, 2008.

[60] Yuan Yu Michael Isard Dennis Fetterly, Mihai Budiu, Úlfar Erlingsson, and Pradeep Kumar Gunda Jon Currey. Dryadlinq: A system for general-purpose distributed data-parallel computing using a high-level language. *Proc. LSDS-IR*, 8, 2009.

[61] John Fitzgerald and Peter Gorm Larsen. *Modelling systems: practical tools and techniques in software development*. Cambridge University Press, 2009.

[62] Anshul Gandhi, Parijat Dube, Alexei Karve, Andrzej Kochut, and Li Zhang. Adaptive, model-driven autoscaling for cloud applications. In *11th International Conference on Autonomic Computing ({ICAC} 14)*, pages 57–64, 2014.

[63] Abolfazl Gandomi, Ali Movaghar, Midia Reshadi, and Ahmad Khademzadeh. Designing a mapreduce performance model in distributed heterogeneous platforms based on benchmarking approach. *The Journal of Supercomputing*, 76(9):7177–7203, 2020.

[64] Ebenezer Komla Gavua and Gabor Kecskemeti. Improving mapreduce speculative executions with global snapshots. *International Journal of Advanced Computer Science and Applications*, 14(1), 2023.

[65] Adem Efe Gencer, David Bindel, Emin Gün Sirer, and Robbert van Renesse. Configuring distributed computations using response surfaces. In *Proceedings of the 16th Annual Middleware Conference*, pages 235–246. ACM, 2015.

[66] Hamoun Ghanbari, Bradley Simmons, Marin Litoiu, Cornel Barna, and Gabriel Iszlai. Optimal autoscaling in a iaas cloud. In *Proceedings of the 9th international conference on Autonomic computing*, pages 173–178. ACM, 2012.

[67] Shamsollah Ghanbari and Mohamed Othman. A priority based job scheduling algorithm in cloud computing. *Procedia Engineering*, 50(1):778–785, 2012.

[68] Tanvi Gupta and Supriya P Panda. Clustering validation of clara and k-means using silhouette & dunn measures on iris dataset. In *2019 International conference on machine learning, big data, cloud and parallel computing (COMITCon)*, pages 10–13. IEEE, 2019.

[69] Yuri Gurevich. Evolving algebras: an attempt to discover semantics., 1993.

[70] hadoopilluminated. Hardware and software for hadoop. `https://hadoopilluminated.com/hadoop_illuminated/Hardware_Software.html`, 2022. [Online; accessed June 23, 2022].

[71] Suhel Hammoud, Maozhen Li, Yang Liu, Nasullah Khalid Alham, and Zelong Liu. MRSim: A discrete event based MapReduce simulator. In *Seventh International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, volume 6, pages 2993–2997, August 2010.

[72] Bingsheng He, Wenbin Fang, Qiong Luo, Naga K Govindaraju, and Tuyong Wang. Mars: a mapreduce framework on graphics processors. In *2008 International Conference on Parallel Architectures and Compilation Techniques (PACT)*, pages 260–269. IEEE, 2008.

[73] Namrata Hosamani, Nageshwar Albur, Prajna Yaji, Mohammed Moin Mulla, and DG Narayan. Elastic provisioning of hadoop clusters on openstack private cloud. In *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, pages 1–7. IEEE, 2020.

[74] Michael Huth and Mark Ryan. *Logic in Computer Science: Modelling and reasoning about systems*. Cambridge university press, 2004.

[75] Michael Isard, Mihai Budiu, Yuan Yu, Andrew Birrell, and Dennis Fetterly. Dryad: distributed data-parallel programs from sequential building blocks. In *ACM SIGOPS operating systems review*, volume 41, pages 59–72. ACM, 2007.

[76] Nemouchi Warda Ismahene, Boudouda Souheila, and Zarour Nacereddine. An auto scaling energy efficient approach in apache hadoop. In *2020 International Conference on Advanced Aspects of Software Engineering (ICAASE)*, pages 1–6. IEEE, 2020.

[77] Devki Nandan Jha, Khaled Alwasel, Areeb Alshoshan, Xianghua Huang, Ranesh Kumar Naha, Sudheer Kumar Battula, Saurabh Garg, Deepak Puthal, Philip James, Albert Zomaya, et al. Iotsim-edge: a simulation framework for modeling the behavior of internet of things and edge computing environments. *Software: Practice and Experience*, 50(6):844–867, 2020.

[78] Hai Jin, Shadi Ibrahim, Li Qi, Haijun Cao, Song Wu, and Xuanhua Shi. The mapreduce programming model and implementations. *Cloud Computing: Principles and Paradigms*, pages 373–390, 2011.

[79] Jongtack Jung and Hwangnam Kim. MR-CloudSim: Designing and implementing MapReduce computing model on CloudSim. In *International Conference on ICT Convergence (ICTC)*, pages 504–509. IEEE, October 2012.

[80] Khushboo Kalia and Neeraj Gupta. Analysis of hadoop mapreduce scheduling in heterogeneous environment. *Ain Shams Engineering Journal*, 12(1):1101–1110, 2021.

[81] Gayathri Karthick, Glenford Mapp, Florian Kammueller, and Mahdi Aiash. Modeling and verifying a resource allocation algorithm for secure service migration for commercial cloud systems. *Computational Intelligence*, 38(3):811–828, 2022.

[82] Pradeeban Kathiravelu and Luís Veiga. Sendim for incremental development of cloud networks: Simulation, emulation and deployment integration middleware. In *2016 IEEE International Conference on Cloud Engineering (IC2E)*, pages 143–146. IEEE, 2016.

[83] Ramandeep Kaur and Navtej Singh Ghumman. A survey and comparison of various cloud simulators available for cloud environment. *Int. J. Adv. Res. Comput. Commun. Eng*, 4(5):605–608, 2015.

[84] Gabor Kecskemeti. Dissect-cf: a simulator to foster energy-aware scheduling in infrastructure clouds. *Simulation Modelling Practice and Theory*, 58:188–218, 2015.

[85] R Palson Kennedy and TV Gopal. A MR simulator in facilitating cloud computing. *International Journal of Computer Applications*, 72(5), 2013.

[86] Ateeq Khan, Johannes Hintsch, Gunter Saake, and Klaus Turowski. Variability management in infrastructure as a service: Scenarios in cloud deployment models. In *International Conference on Computing, Networking and Communications (ICNC)*, pages 724–728, January 2017.

[87] Abhishek Khanna and Sanmeet Kaur. Internet of things (iot), applications and challenges: a comprehensive review. *Wireless Personal Communications*, 114(2):1687–1762, 2020.

[88] Dzmitry Kliazovich, Pascal Bouvry, and Samee Ullah Khan. Greencloud: a packet-level simulator of energy-aware cloud computing data centers. *The Journal of Supercomputing*, 62(3):1263–1283, 2012.

[89] Wagner Kolberg, Pedro De B Marcos, Julio CS Anjos, Alexandre KS Miyazaki, Claudio R Geyer, and Luciana B Arantes. Mrsg–a mapreduce simulator over simgrid. *Parallel Computing*, 39(4-5):233–244, 2013.

[90] Maria Krotsiani, Christos Kloukinas, and George Spanoudakis. Cloud certification process validation using formal methods. In *International Conference on Service-Oriented Computing*, pages 65–79. Springer, 2017.

[91] TKS LakshmiPriya and Ranjani Parthasarathi. An asm model for an autonomous network-infrastructure grid. In *International Conference on Networking and Services (ICNS'07)*, pages 29–29. IEEE, 2007.

[92] Ralf Lämmel. Google's mapreduce programming model—revisited. *Science of computer programming*, 70(1):1–30, 2008.

[93] Sara Landset, Taghi M Khoshgoftaar, Aaron N Richter, and Tawfiq Hasanin. A survey of open source tools for machine learning with big data in the hadoop ecosystem. *Journal of Big Data*, 2(1):24, 2015.

[94] Chia-Wei Lee, Kuang-Yu Hsieh, Sun-Yuan Hsieh, and Hung-Chang Hsiao. A dynamic data placement strategy for hadoop in heterogeneous environments. *Big Data Research*, 1:14–22, 2014.

[95] Daewoo Lee, Jin-Soo Kim, and Seungryoul Maeng. Large-scale incremental processing with mapreduce. *Future Generation Computer Systems*, 36:66–79, 2014.

[96] In Lee and Kyoochun Lee. The internet of things (IoT): Applications, investments, and challenges for enterprises. *Business Horizons*, 58(4):431–440, 2015.

[97] Jianjiang Li, Yajun Liu, Jian Pan, Peng Zhang, Wei Chen, and Lizhe Wang. Map-balance-reduce: An improved parallel programming model for load balancing of mapreduce. *Future Generation Computer Systems*, 105:993–1001, 2020.

[98] Ren Li, Haibo Hu, Heng Li, Yunsong Wu, and Jianxi Yang. Mapreduce parallel programming model: A state-of-the-art survey. *International Journal of Parallel Programming*, 44(4):832–866, 2016.

[99] Yuanzhen Li, Qun Yang, Shangqi Lai, and Bohan Li. A new speculative execution algorithm based on c4. 5 decision tree for hadoop. In *International Conference of Young Computer Scientists, Engineers and Educators*, pages 284–291. Springer, 2015.

[100] Seung-Hwan Lim, Bikash Sharma, Gunwoo Nam, Eun Kyoung Kim, and Chita R Das. Mdcsim: A multi-tier data center simulation, platform. In *2009 IEEE International Conference on Cluster Computing and Workshops*, pages 1–9. IEEE, 2009.

[101] Jia-Chun Lin, Ingrid Chieh Yu, Einar Broch Johnsen, and Ming-Chang Lee. Abs-yarn: A formal framework for modeling hadoop yarn clusters. In *International Conference on Fundamental Approaches to Software Engineering*, pages 49–65. Springer, 2016.

[102] Xiaocheng Liu, Xiaogang Qiu, Bin Chen, and Kedi Huang. Cloud-based simulation: The state-of-the-art computer simulation paradigm. In *2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation*, pages 71–74. IEEE, 2012.

[103] Yang Liu, Maozhen Li, Nasullah Khalid Alham, and Suhel Hammoud. Hsim: a mapreduce simulator in enabling cloud computing. *Future Generation Computer Systems*, 29(1):300–308, 2013.

[104] Tania Lorido-Botran, Jose Miguel-Alonso, and Jose A Lozano. A review of auto-scaling techniques for elastic applications in cloud environments. *Journal of grid computing*, 12(4):559–592, 2014.

[105] Adil Maarouf, Abderrahim Marzouk, and Abdelkrim Haqiq. Comparative study of simulators for cloud computing. In *2015 International Conference on Cloud Technologies and Applications (CloudTech)*, pages 1–8. IEEE, 2015.

[106] Sulav Malla and Ken Christensen. Hpc in the cloud: Performance comparison of function as a service (faas) vs infrastructure as a service (iaas). *Internet Technology Letters*, 3(1):e137, 2020.

[107] Najme Mansouri, R Ghafari, and B Mohammad Hasani Zade. Cloud computing simulators: A comprehensive review. *Simulation Modelling Practice and Theory*, 104:102144, 2020.

[108] Sunilkumar S Manvi and Gopal Krishna Shyam. Resource management for infrastructure as a service (IaaS) in cloud computing: A survey. *Journal of Network and Computer Applications*, 41:424–440, 2014.

[109] Andras Markus, Mate Biro, Gabor Kecskemeti, and Attila Kertesz. Actuator behaviour modelling in iot-fog-cloud simulation. *PeerJ Computer Science*, 7:e651, 2021.

[110] Federico Milano. Drawbacks of using simplified models to teach power system analysis. In *EDULEARN20 Proceedings*, pages 121–129. IATED, 2020.

[111] Francesco Moscato. Exploiting model profiles in requirements verification of cloud systems. *Int. J. High Perform. Comput. Netw.*, 8(3):259–274, 2015.

[112] Vasileios Moysiadis, Panagiotis Sarigiannidis, and Ioannis Moscholios. Towards distributed data management in fog computing. *Wireless Communications and Mobile Computing*, 2018, 2018.

[113] Prashanth Mundkur, Ville Tuulos, and Jared Flatow. Disco: a computing platform for large-scale data analytics. In *Proceedings of the 10th ACM SIGPLAN workshop on Erlang*, pages 84–89. ACM, 2011.

[114] Arun C Murthy. Mumak: Map-reduce simulator. techreport MAPREDUCE-728, Apache Software Foundation, July 2009.

[115] Zsolt Németh and Vaidy Sunderam. Characterizing grids: Attributes, definitions, and formalisms. *Journal of Grid Computing*, 1:9–23, 2003.

[116] Warda Ismahene Nemouchi, Souheila Boudouda, and Nacer Eddine Zarour. Efficient auto scaling and cost-effective architecture in apache hadoop. In *International Conference on Artificial Intelligence and its Applications*, pages 336–345. Springer, 2022.

[117] Marco AS Netto, Carlos Cardonha, Renato LF Cunha, and Marcos D Assunçao. Evaluating auto-scaling strategies for cloud computing environments. In *2014 IEEE 22nd International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems*, pages 187–196. IEEE, 2014.

[118] Leonardo Neumeyer, Bruce Robbins, Anish Nair, and Anand Kesari. S4: Distributed stream computing platform. In *IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 170–177, December 2010.

[119] Alberto Núñez, Pablo C Cañizares, and Juan de Lara. Cloudexpert: An intelligent system for selecting cloud system simulators. *Expert Systems with Applications*, page 115955, 2021.

[120] packtpub. Sizing and configuring your hadoop cluster. `https://hub.packtpub.com/sizing-configuring-hadoop-cluster/`, 2022. [Online; accessed April 23, 2023].

[121] Jianli Pan and Raj Jain. A survey of network simulation tools: Current status and future developments. techreport, Washington University in St. Louis, 2008.

[122] Biswanath Panda, Joshua S Herbach, Sugato Basu, and Roberto J Bayardo. Planet: massively parallel learning of tree ensembles with mapreduce. *Proceedings of the VLDB Endowment*, 2(2):1426–1437, 2009.

[123] Junjie Peng, Xuejun Zhang, Zhou Lei, Bofeng Zhang, Wu Zhang, and Qing Li. Comparison of several cloud computing platforms. In *Second International Symposium on Information Science and Engineering (ISISE 2009)*, pages 23–27. IEEE, 2009.

[124] Project Pro. Top 10 industries using big data and 121 companies who hire hadoop developers. `https://www.projectpro.io/article/top-10-industries-using-big-data`, 2023. [Online; accessed April 23, 2023].

[125] Ramakrishnan Ramanathan and B Latha. Towards optimal resource provisioning for hadoop-mapreduce jobs using scale-out strategy and its performance analysis in private cloud environment. *Cluster Computing*, 22(6):14061–14071, 2019.

[126] Dimpi Rani and Rajiv Kumar Ranjan. A comparative study of saas, paas and iaas in cloud computing. *International Journal of Advanced Research in Computer Science and Software Engineering*, 4(6), 2014.

[127] Alexander Rasmussen, Michael Conley, George Porter, Rishi Kapoor, Amin Vahdat, et al. Themis: an i/o-efficient mapreduce. In *Proceedings of the Third ACM Symposium on Cloud Computing*, page 13. ACM, 2012.

[128] Aysan Rasooli and Douglas G Down. An adaptive scheduling algorithm for dynamic heterogeneous hadoop systems. In *Proceedings of the 2011 Conference of the Center for Advanced Studies on Collaborative Research*, pages 30–44. IBM Corp., 2011.

[129] Zujie Ren, Zhijun Liu, Xianghua Xu, Jian Wan, Weisong Shi, and Min Zhou. Waxelephant: A realistic hadoop simulator for parameters tuning and scalability analysis. In *Seventh ChinaGrid Annual Conference*, pages 9–16, September 2012.

[130] Markus Roggenbach, Antonio Cerone, Bernd-Holger Schlingloff, Gerardo Schneider, and Siraj Ahmed Shaikh. Formal methods for software engineering–languages, methods, application domains, 2020.

[131] Hamza Sahli, Faïza Belala, and Chafia Bouanaka. Formal verification of cloud systems elasticity. *International Journal of Critical Computer-Based Systems*, 6(4):364–384, 2016.

[132] Sherif Sakr, Anna Liu, and Ayman G Fayoumi. The family of mapreduce and large-scale data processing systems. *ACM Computing Surveys (CSUR)*, 46(1):1–44, 2013.

[133] Mario Santana. Infrastructure as a service (iaas). In *Cloud Computing Security*, pages 65–70. CRC Press, 2020.

[134] Deepika Saxena and Ashutosh Kumar Singh. A proactive autoscaling and energy-efficient vm allocation framework using online multi-resource neural network for cloud data center. *Neurocomputing*, 426:248–264, 2021.

[135] Gerhard Schellhorn. Asm refinement and generalizations of forward simulation in data refinement: A comparison. *Theoretical Computer Science*, 336(2-3):403–435, 2005.

[136] Hermes Senger, Veronica Gil-Costa, Luciana Arantes, Cesar AC Marcondes, Mauricio Marin, Liria M Sato, and Fabrício AB Da Silva. Bsp cost and scalability analysis for mapreduce operations. *Concurrency and Computation: Practice and Experience*, 28(8):2503–2527, 2016.

[137] Utkal Sinha and Mayank Shekhar. Comparison of various cloud simulation tools available in cloud computing. *International Journal of Advanced Research in Computer and Communication Engineering*, 4(3):171–176, 2015.

[138] Lara Kathleen Smetana and Randy L Bell. Computer simulations to support science instruction and learning: A critical review of the literature. *International Journal of Science Education*, 34(9):1337–1370, 2012.

[139] Cagatay Sonmez, Atay Ozgovde, and Cem Ersoy. Edgecloudsim: An environment for performance evaluation of edge computing systems. *Transactions on Emerging Telecommunications Technologies*, 29(11):e3493, 2018.

[140] Alireza Souri. Formal specification and verification of a data replication approach in distributed systems. *Int J Next Gener Comput*, 7(1):18–37, 2016.

[141] Alireza Souri, Monire Norouzi, Adalat Safarkhanlou, and Seyed Hassan Es Haghi Sardroud. A dynamic data replication with consistency approach in data grids: modeling and verification. *Baltic Journal of Modern Computing*, 4(3):546, 2016.

[142] Confluence Spaces. Companies that offer services on or based around hadoop. `https://cwiki.apache.org/confluence/display/HADOOP2/poweredby#PoweredBy-S`, 2022. [Online; accessed April 23, 2023].

[143] Georgios L Stavrinides and Helen D Karatza. The impact of data locality on the performance of a saas cloud with real-time data-intensive applications. In *Proceedings of the 21st International Symposium on Distributed Simulation and Real Time Applications*, pages 180–187, 2017.

[144] Michael Stonebraker, Uğur Çetintemel, and Stan Zdonik. The 8 requirements of real-time stream processing. *ACM Sigmod Record*, 34(4):42–47, 2005.

[145] Mingming Sun, Hang Zhuang, Changlong Li, Kun Lu, and Xuehai Zhou. Scheduling algorithm based on prefetching in mapreduce clusters. *Applied Soft Computing*, 38:1109–1118, 2016.

[146] Xiaoyu Sun, Chen He, and Ying Lu. Esamr: An enhanced self-adaptive mapreduce scheduling algorithm. In *2012 IEEE 18th International Conference on Parallel and Distributed Systems*, pages 148–155. IEEE, 2012.

[147] Sayantan Sur, Hao Wang, Jian Huang, Xiangyong Ouyang, and Dhabaleswar K Panda. Can high-performance interconnects benefit hadoop distributed file system. In *Workshop on Micro Architectural Support for Virtualization, Data Center Computing, and Clouds (MASVDC). Held in Conjunction with MICRO*, page 10. Citeseer, 2010.

[148] Pericherla S Suryateja. A comparative analysis of cloud simulators. *International Journal of Modern Education & Computer Science*, 8(4), 2016.

[149] Tomasz Szydlo, Amadeusz Szabala, Nazar Kordiumov, Konrad Siuzdak, Lukasz Wolski, Khaled Alwasel, Fawzy Habeeb, and Rajiv Ranjan. Iotsim-osmosis-res: Towards autonomic renewable energy-aware osmotic computing. *Software: Practice and Experience*, 2022.

[150] Justin Talbot, Richard M Yoo, and Christos Kozyrakis. Phoenix++: modular mapreduce for shared-memory systems. In *Proceedings of the second international workshop on MapReduce and its applications*, pages 9–16. ACM, 2011.

[151] Mehdi Tavakolan, Farzad Mostafazadeh, Saeed Jalilzadeh Eirdmousa, Amir Safari, and Kaveh Mirzaei. A parallel computing simulation-based multi-objective optimization framework for economic analysis of building energy retrofit: A case study in iran. *Journal of Building Engineering*, 45:103485, 2022.

[152] Fei Teng, Lei Yu, and Frederic Magoulès. SimMapReduce: A simulator for modeling mapreduce framework. In *5th FTRA International Conference on Multimedia and Ubiquitous Engineering (MUE)*, pages 277–282, June 2011.

[153] Akhilesh S Thyagaturu, Prateek Shantharama, Ahmed Nasrallah, and Martin Reisslein. Operating systems and hypervisors for network functions: A survey of enabling technologies and research studies. *IEEE Access*, 2022.

[154] Fengguang Tian and Keke Chen. Towards optimal resource provisioning for running mapreduce programs in public clouds. In *IEEE International Conference on Cloud Computing (CLOUD)*, pages 155–162, July 2011.

[155] titanwolf. Configuring your hadoop cluster. `https://titanwolf.org/Network/Articles/Article?AID=d8fe9daf-88cf-48f9-9e89-99cccef6f69d`, 2022. [Online; accessed June 23, 2022].

[156] Qazi Zia Ullah, Gul Muhammad Khan, and Shahzad Hassan. Cloud infrastructure estimation and auto-scaling using recurrent cartesian genetic programming-based ann. *IEEE Access*, 8:17965–17985, 2020.

[157] Ramazan Ünlü and Petros Xanthopoulos. Estimating the number of clusters in a dataset via consensus clustering. *Expert Systems with Applications*, 125:33–39, 2019.

[158] Wil Van der Aalst, Arya Adriansyah, and Boudewijn van Dongen. Replaying history on process models for conformance checking and performance analysis. *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, 2(2):182–192, 2012.

[159] Abhishek Verma, Ludmila Cherkasova, and Roy H Campbell. Play it again, simmr! In *2011 IEEE International Conference on Cluster Computing*, pages 253–261. IEEE, 2011.

[160] V V Vinothina, R Sridaran, and Padmavathi Ganapathi. A survey on resource allocation strategies in cloud computing. *International Journal of Advanced Computer Science and Applications*, 3(6), 2012.

[161] w3trainingschool. Top companies using apache hadoop, 2022.

[162] Guanying Wang. *Evaluating mapreduce system performance: A simulation approach*. PhD thesis, Virginia Tech, 2012.

[163] Guanying Wang, Ali R Butt, Henry Monti, and Karan Gupta. Towards synthesizing realistic workload traces for studying the hadoop ecosystem. In *IEEE 19th International Symposium on Modelling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 400–408, July 2011.

[164] Guanying Wang, Ali R Butt, Prashant Pandey, and Karan Gupta. A simulation approach to evaluating design decisions in mapreduce setups. In *IEEE International Symposium on Modeling, Analysis & Simulation of Computer and Telecommunication Systems (MASCOTS).*, pages 1–11, September 2009.

[165] Guanying Wang, Ali R Butt, Prashant Pandey, and Karan Gupta. Using realistic simulation for performance analysis of mapreduce setups. In *Proceedings of the 1st ACM workshop on Large-Scale system and application performance*, pages 19–26. ACM, 2009.

[166] Md Wasi-Ur-Rahman, Nusrat Sharmin Islam, Xiaoyi Lu, Dipti Shankar, and Dhabaleswar K Panda. MR-Advisor: A comprehensive tuning tool for advising HPC users to accelerate MapReduce applications on supercomputers. In *28th International Symposium on Computer Architecture and High Performance Computing (SBAC-PAD)*, pages 198–205, October 2016.

[167] Tom White. *Hadoop: The definitive guide*. O'Reilly Media, Inc., 2012.

[168] Bhathiya Wickremasinghe, Rodrigo N Calheiros, and Rajkumar Buyya. Cloudanalyst: A cloudsim-based visual modeller for analysing cloud computing environments and applications. In *2010 24th IEEE international conference on advanced information networking and applications*, pages 446–452. IEEE, 2010.

[169] Frederik Wulf. Platform-as-a-service post-adoption by large-scale organizations: A research model design. 2020.

[170] Huanle Xu and Wing Cheong Lau. Speculative execution for a single job in a mapreduce-like system. In *2014 IEEE 7th International Conference on Cloud Computing*, pages 586–593, 2014.

[171] Jie Xu, Edward Huang, Chun-Hung Chen, and Loo Hay Lee. Simulation optimization: A review and exploration in the new era of cloud computing and big data. *Asia-Pacific Journal of Operational Research*, 32(03):1550019, 2015.

[172] Haibo Yang and Mary Tate. A descriptive literature review and classification of cloud computing research. *CAIS*, 31:2, 2012.

[173] Hailong Yang, Zhongzhi Luan, Wenjun Li, and Depei Qian. Mapreduce workload modeling with statistical approach. *Journal of grid computing*, 10(2):279–310, 2012.

[174] Jingqi Yang, Chuanchang Liu, Yanlei Shang, Zexiang Mao, and Junliang Chen. Workload predicting-based automatic scaling in service clouds. In *2013 IEEE Sixth International Conference on Cloud Computing*, pages 810–815. IEEE, 2013.

[175] Z Yao-cheng. *General purpose parallel discrete event simulation environment and the study of relevant techniques*. PhD thesis, PhD Thesis. National University of Defense Technology, Changsha, Hunan, PR China, 2008.

[176] Awad A Younis, Rajshekhar Sunderraman, Mike Metzler, and Anu G Bourgeois. Developing parallel programming and soft skills: A project based learning approach. *Journal of Parallel and Distributed Computing*, 158:151–163, 2021.

[177] Chunhui Yuan and Haitao Yang. Research on k-value selection method of k-means clustering algorithm. *J*, 2(2):226–235, 2019.

[178] Matei Zaharia, Mosharaf Chowdhury, Michael J Franklin, Scott Shenker, and Ion Stoica. Spark: Cluster computing with working sets. *HotCloud*, 10(10-10):95, 2010.

[179] Matei Zaharia, Andy Konwinski, Anthony D Joseph, Randy H Katz, and Ion Stoica. Improving mapreduce performance in heterogeneous environments. In *Osdi*, volume 8, page 7, 2008.

[180] Xuezhi Zeng, Saurabh Kumar Garg, Peter Strazdins, Prem Prakash Jayaraman, Dimitrios Georgakopoulos, and Rajiv Ranjan. IOTSim: A

simulator for analysing iot applications. *Journal of Systems Architecture*, 72:93–107, 2017.

[181] Wei Zhao, Yong Peng, Feng Xie, and Zhonghua Dai. Modeling and simulation of cloud computing: A review. In *2012 IEEE Asia Pacific cloud computing congress (APCloudCC)*, pages 20–24. IEEE, 2012.

[182] Hong Bo Zhou and Jun Tao Gao. Automatic method for determining cluster number based on silhouette coefficient. In *Advanced materials research*, volume 951, pages 227–230. Trans Tech Publ, 2014.