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

Table of Contents

1 Introduction

Nowadays, the field of cloud computing is experiencing an increased rate in research focused on resource provisioning frameworks [7]. These works have encouraged the utilisation 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). Researchers and organisations utilise MR for applications that process large data sets in parallel on clusters [4].

MapReduce has several implementations designed for specific purposes. One of such implementation developed for distributed storage and processing large datasets is apache hadoop. The core of Hadoop includes the Hadoop Distributed File System (HDFS) and a MR processor [5]. Hadoop executes MR programs written in different languages. In spite of all the benefits derived from Hadoop, it has few challenges. One of such challenges is how it deals with tasks which require abnormally long run time. MR reprocesses unusually long tasks (straggler tasks) on available nodes to finish the computation faster (as backup tasks) [4]. This phenomenon is known as speculative execution. Moreover, to ensure that challenges on clouds are tackled with less efforts [1], researchers have resorted to the use of simulations.

Several benefits have been derived from the application of cloud simulations. Since these simulations foster the provisions of requisite computational capacity to solve problems in a reasonable amount of time [9]. Over time, research has progressed to evaluating computer simulations tools to highlight their benefits and disadvantages. Mansouri et al [7] reviewed over thirty-three cloud simulators based on multi-level feature analysis of simulators. Byrne et al [2] reviewed thirty-three simulators based on autonomous simulation platforms. Amongst the simulators reviewed includes CloudSim [3] and DISSECT-CF [6]. One of the main benefits of utilising simulations is the provisioning of resources to meet demands. This is reflective of what pertains in auto-scaling mechanisms (auto-scalers).

Several auto-scaling research efforts have been carried out to analyse the resource provisioning behaviours exhibited by auto-scalers; especially when they emanate from different cloud infrastructures. Most of these efforts applied statistical and experimental [10] approaches. However, research has shown that there are available flexible and verifiable ways to evaluate the resource provisioning behaviours of auto-scalers [8].

Now, the issues presented above require a strategic direction 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 MR 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.

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.

2 Background

2.1 Cloud Simulation

Computer Simulation (CS) has become an important tool for design, analysis, and evaluating systems, and has been playing important roles in several domains including economy, medicine and entertainment, with great success. The main benefits of cloud simulators are cost minimization and repeatable and controllable experiments. Some of the few drawbacks of CS are simplifications and Model validation [9]. The effects of these drawbacks should be minimized to ensure the usefulness of simulation results.

2.2 Abstract State Machine Theory

The ASM theory encompasses a formal system engineering technique that guides software development. It begins from software requirements capture to their implementation. This theory is implemented via the ASM ground moddel. A *ground model* is an ASM that can be considered a rigorous high-level system blueprint. It is mostly developed via a model refinement process. A *model refinement* is a general scheme allows progression from an abstract model to a more detailed one [8]. This process is closely linked with the ASM refinement method and *Börger's refinement*. Now, let us discuss the design of our *Mareclass* framework.

3 MapReduce Simulators Selection Framework Design and Implementation

3.1 Overview of the Classification Framework

In order to analyse the existing MR cloud simulation and modelling tools, a classification (*MaReClass*) framework was proposed. Our *MaReClass* framework consists of a set of criteria used to evaluate and classify several MR simulators to highlight their strengths and weaknesses. The criteria in the framework were derived from the functional requirements of simulators. *MaReClass* 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. 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

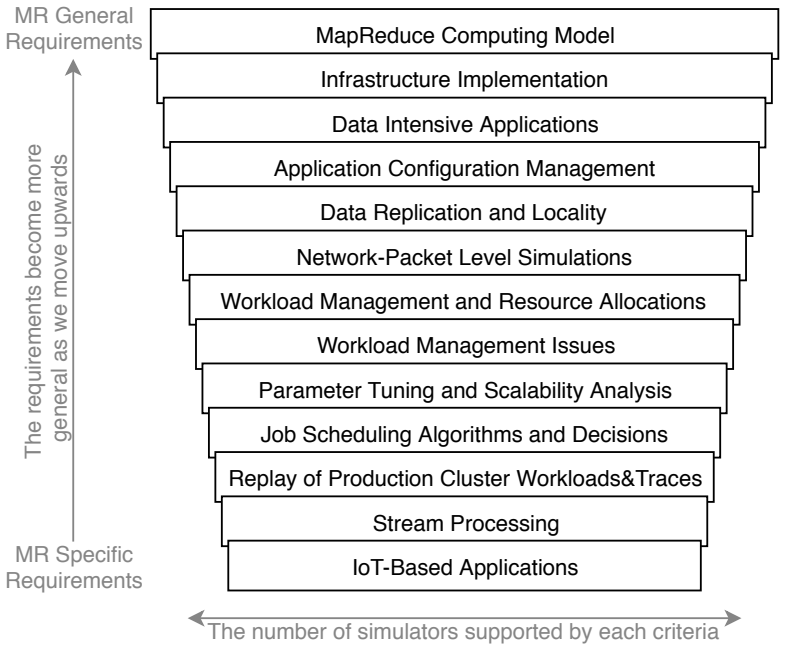


Figure 1: Criteria for Classification of MapReduce Simulators

thirty themes to thirteen themes most relevant to MR simulations as shown in figure 1. The refining process focused on the most cited themes amongst the initial list. 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.

3.2 MapReduce Simulators Classification framework

Our *MaReClass* framework displayed in figure 1 was applied to evaluate selected MR simulators as shown in figure 2.

Our analysis has shown that each of the MR simulators support MR features to some extent. Therefore, from figure 2 we can see that aside MR-Cloudsim and MrPerf, all the other simulators support most of the features required for conducting MR research with percentage representation above 50%. Therefore, it is recommended that simulator developers focus on incorporating some of these unavailable features into their works. Let us now discuss the design and implementation of a *Haspeck* solution on MRH.

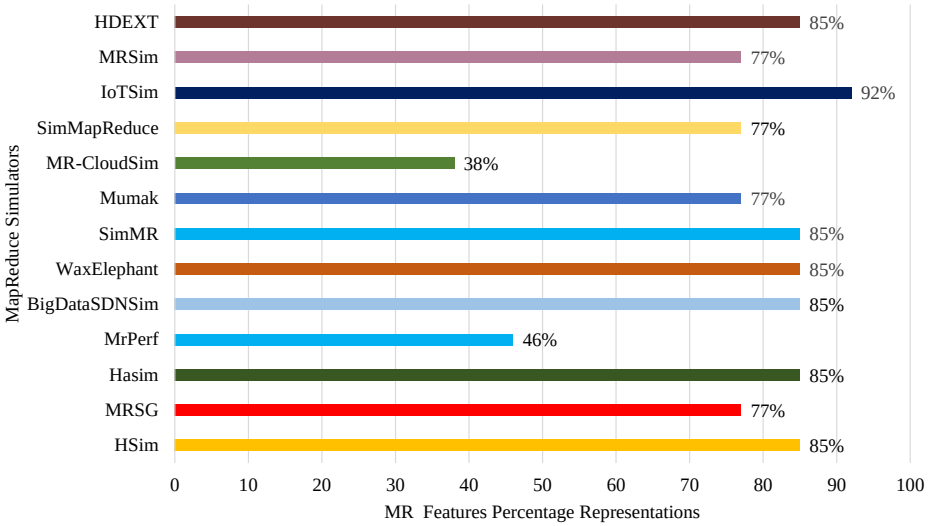


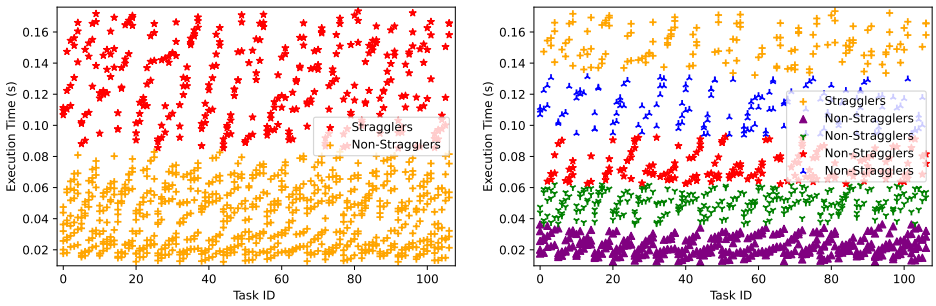
Figure 2: Percentage Representation of MR Simulators Support for Specific Features

4 Design and Implementation of *Haspeck* Solution on MapReduce Hadoop

4.1 Design of *Haspeck* Solution

Haspeck consists of three algorithms designed to dynamically collects real-time data from all types of environments. These algorithms are interconnected to ensure the determination of task run times and appropriate selection of backup tasks. The algorithms are the snapshots capturing, task performance and task instance monitoring algorithms. K-means clustering algorithm is implemented with silhouette scores to categorise task run times (data set) received from the snapshot capturing algorithm. The data set are classified into straggler and non-straggler tasks.

Furthermore, the K-means algorithm was implemented in our work as a decision-making tool. There are cases where the data set 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 as seen in figures 3a to 3b. As such, silhouette scores was employed to validate well-defined data clusters as seen in figures 4a to 4b; which will require rescheduling of the straggler tasks.



(a) 2-k Data Clusters **(b)** 5-k Data Clusters

Figure 3: Kmeans Data Clusters of Tasks Execution times from 20 Nodes by 8 Cores Data Centre with No-Disruption

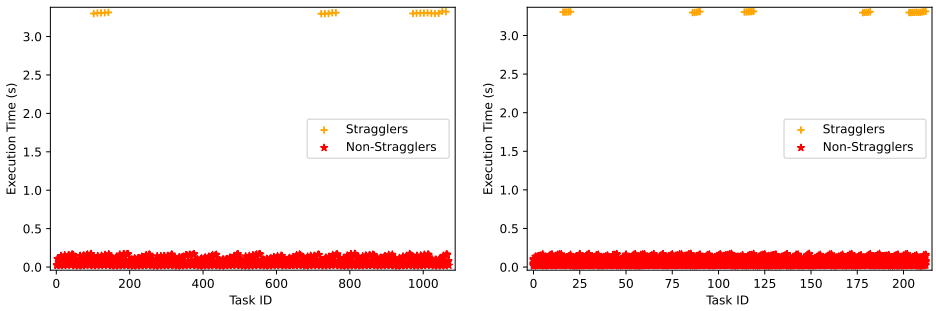
Table 1: KMeans Clustering Silhouette Scores of No-Disrupted Data Clusters

Figure Number	Silhouette Scores
3a	0.685
3b	0.615
4a	0.985
4b	0.985

4.2 Evaluation of *Haspeck*

Our *Haspeck* solution was assessed through three major experiments to prove its applicability. They are (*i*) strategy implementation overheads (*ii*) job performance (*iii*) Evaluation with Baseline Methods. The experiments enabled us to see the applicability of our approach. The experiment was conducted on our extension of HDMSG MapReduce (a MapReduce simulator with Simgrid as the main backbone) available on GitHub [1]. 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. To determine real life MR cluster infrastructure and application configurations, two surveys about hadoop cluster requirements were carried out. The survey focused on identifying typical hadoop configurations and organisations actively utilising hadoop clusters. The findings of the survey fostered the selection of four infrastructure scenarios (displayed in table 2) for our experiments.

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



(a) 20 nodes by 8 cores **(b)** 100 nodes by 8 cores

Figure 4: Kmeans Clustering of Run Times from Experimental Scenarios

Table 2: 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

4.3 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. The experiment was conducted on the four data centre scenarios discussed in sub-section 4.2. Mapper tasks with execution times from 0.5 to 2000 seconds were utilized and we scaled the executions until the graph converged at 2000 seconds. Since *Haspeck* involves capturing snapshots during the processing of smaller mappers (as checkpoint barriers), measurements were taken and utilised to determine the overheads on a single mapper. The overheads of a single mapper were measured on the four data centre scenarios are seen in figure 5. Let us discuss the results of the overheads.

First, the impact of applying *Haspeck* was gradual on the 20N×8c scenario. The overheads were high at the initial stages of the experiment. How-

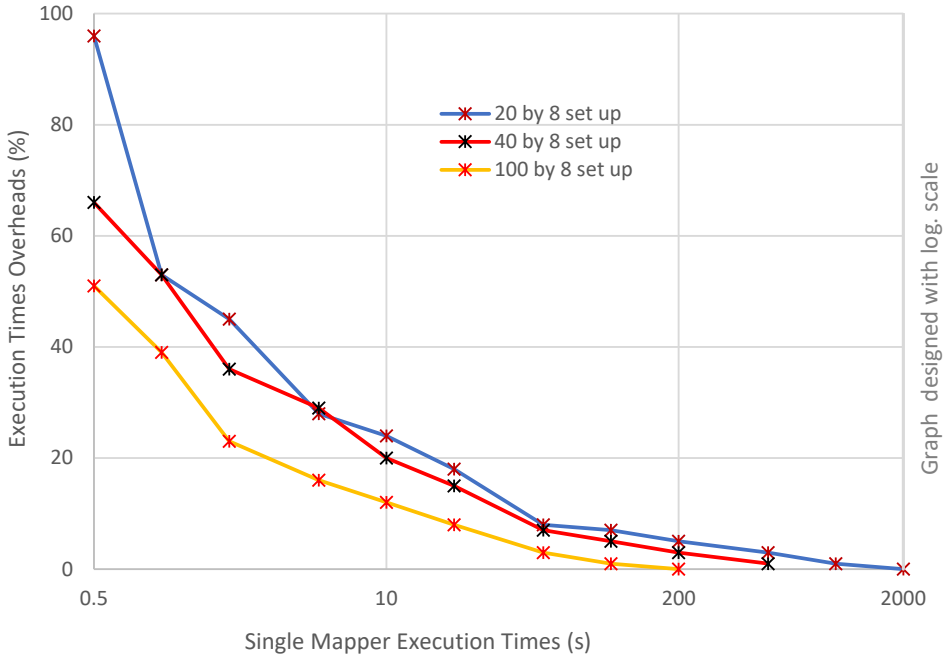


Figure 5: Cluster Overheads

ever, the overheads reduced gradually with longer run times, as seen in figure 5. Second, the 20N×16c and 40N×8c scenarios exhibited similar overhead behaviours during task runs. Therefore, only the 40N×8c set up was shown in figure 5. The initial overheads observed were 1.5% lower (relatively) than the 20N×8c data centre scenario. Third, the 100N×8c scenario demonstrates how *Haspeck* deals with larger data-centres. 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. In conclusion, the overheads reduce faster with long mapper run times.

4.4 Job Performance Experiments

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

- Total execution times when there was *no-disruption* on the MapReduce set-up (i.e., a dedicated hadoop cluster scenario).

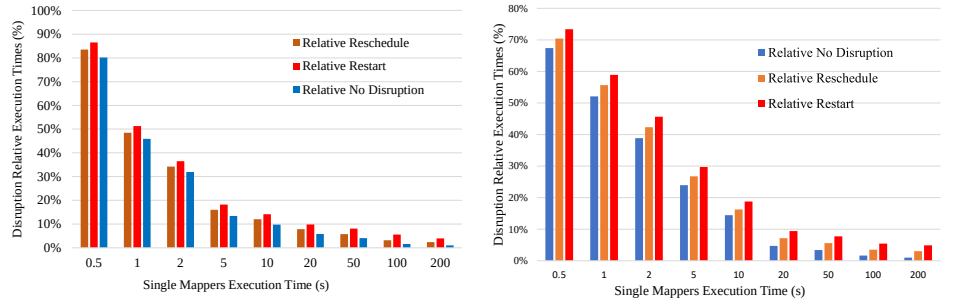
- Total execution times when *disruptions* were introduced on arbitrarily nodes on the infrastructure.
- Total execution times when tasks were terminated and processed as backup tasks *resuming* from their snapshots on a different host (this represents situations when mappers can restore their mid-execution states) (i.e. *reschedule* backup tasks).
- Total execution times when tasks were terminated and processed as backup tasks *restart* on a different host.

These measurements were utilized to draw the graphs shown in figure 6 relative to the disruption introduced. This was done to show the job performance improvements compared to the disruptions. The disruption introduced is at the 100% mark on figure 6. The effects of the disruption is 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 scenarios are discussed as follows.

First, figure 6a shows the behaviour of scenario 20N×8c data centre. The job performance improvement slope began from above 80% at 0.5 seconds and continued to below 40% at 5 seconds. *Haspeck* improved from below 20% at 10 seconds to below 5% at 200 seconds relative to disruptions.

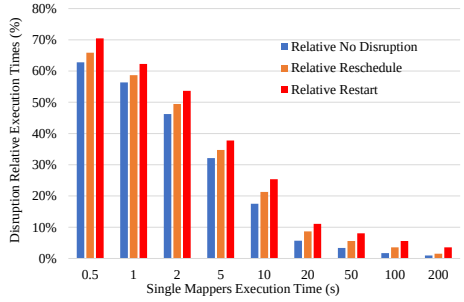
Second, scenario 20N×16c data centre (figure 6b) demonstrated considerably improvement compared to the 20N×8c data centre as the bar graphs were below the 80% mark. The job performance improvement slope began from above 73% at 0.5 seconds to below 40% at 5 seconds. *Haspeck* improved to below 20% at 20 seconds. Third, scenario 40N×8c data centre improved more compared to the previous two previous scenarios as seen in figure 6c. In relation to the *disruption* graph, the job improvement began from above 70% at 0.5 seconds to below 40% at 5 seconds. *Haspeck* improved to 5% and below at 200 seconds. Fourth, The 100N×8c (figure 6d) scenario improved more than all the previous three scenarios. The graph showed a better improvement from below the 60% at 0.5 seconds. At 5 seconds, *Haspeck* improved to 40%. The job performance improvement continued to 2% at 200 seconds. The figure showed that jobs with long run times had higher chances of improvement in this data centre. In general, 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.

Futhermore, some baseline methods were executed on our experimental set-up and the results compared *Haspeck*. The methods were the Hadoop Naive, Longest Approximate Time To End (LATE) and the Self-Adaptive

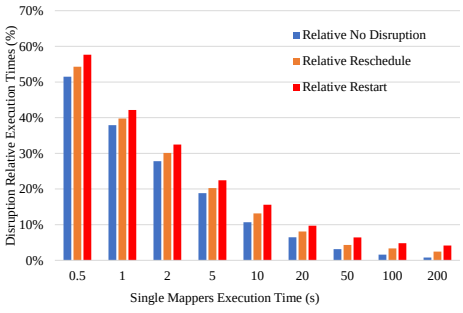


(a) 20 nodes by 8 cores

(b) 20 nodes by 16 cores



(c) 40 nodes by 8 cores



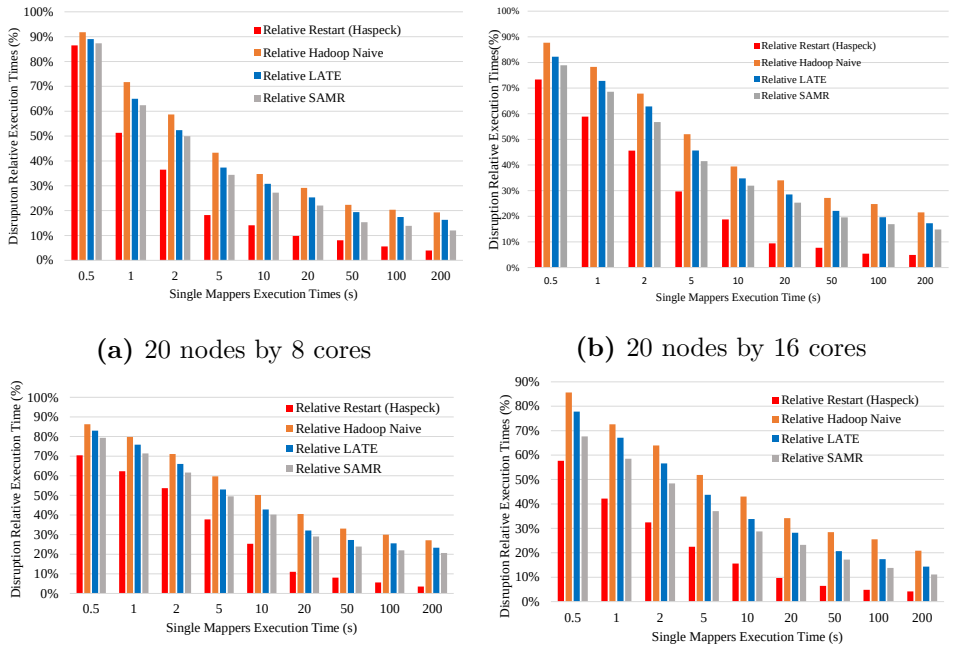
(d) 100 nodes by 8 cores

Figure 6: Jobs Improvement Experimental Scenarios (Drawn Relative to *Disruptions*)

MR Scheduling Algorithms (SAMR) [4]. Appropriate precautions were taken to ensure that the values produced for analysis were accurate.

4.5 Experimental Discussion

The experiments of the baseline methods were carried out on the four data centre scenarios discussed in sub-section 4.4. The measurements were compared with *Haspeck*. Also, the graphs of all the methods were drawn relative to disruptions as seen in figure 7. Let us now highlight the job improvements observed. First, on the 20N×8c data centre configuration, our approach showed 1.67, 1.51 and 1.39 times average job improvements over Hadoop Naive, LATE and SAMR as seen in figure 7a. Second, on the 20N×16c data centre configuration, our approach showed 1.70, 1.52 and 1.40 times average job improvements over Hadoop Naive, LATE and SAMR as seen in figure 7b. 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 as seen in figure 7c. Fourth, the 100N×8c data centre configuration



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

showed 2.18, 1.84 and 1.56 times average job improvements over Hadoop Naive, LATE and SAMR as seen in figure 7d.

4.6 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 4a and 4b. To determine the number of k-clusters suitable for our work, we generated several clusters from our experimental data set. A visualization of some of the data clusters (k=2 to k=5) for both categories is seen in figures 3. The silhouette scores of figures 3a and 3b show a reduction in value as the number of data clusters increased as seen in table 1. These values are closer to our silhouette score threshold lower-bound value. This is due to the fact that the euclidean distance between the centroids decreases as the number of data clusters increase. This affects the decision making of the silhouette scores. Nevertheless, the time bound for all data clusters are almost the same.

5 Modelling Auto-Scaling Mechanisms in Clouds

5.1 Investigating the behaviours of Auto-Scaling Mechanisms

The design of our *Astam* Model required the establishment of a ground model. However, we needed to investigate the behaviours of auto-scalers on DISSECT-CF and literature as an inspiration. This helped to identify shared components and their behaviours. These components were documented to aid the design of *Astam*. Once the ground model was established, it was validated together with its refinements to ascertain its applicability to other auto-scalers.

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. In this research, the simulator's infrastructure management system is the main focus. In building our investigations, the simulator's auto-scaling related examples[2] were examined. The specific actions taken include:

- The source code was observed before and also at run time about how the algorithms are put together.
- The auto-scaling part was extracted out from the rest and presented.
- Based on the observation, two extra auto-scalers were created.
- These are presented here as the model.

The auto-scalers were built on several components presented in figure 8. 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. The auto-scalers offered by DISSECT-CF are ThresholdBasedVI (Threshold), VMCreationPriorityBasedVI (Vmcreate), PoolingVI Mechanism (Pooling), VMOptimisationBased (Vmopt) and FixedVM (Fixed) as seen in figure 8. Now, let us discuss the design of our *Astam* model.

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

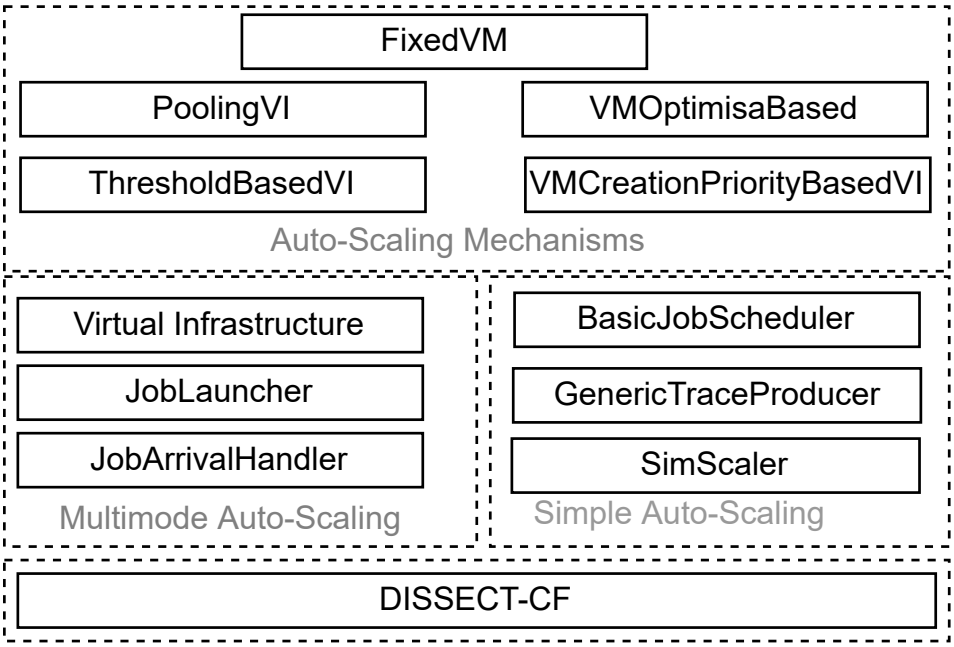


Figure 8: Architectural view of Auto-Scaling Mechanisms on DISSECT-CF.

5.2 Design of an Astam Model

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

Step 1: Design and Analysis of the model's framework as displayed in figures 9 and 11. The framework shows our model's ground model for the two categories of auto-scalers. Figure 9 shows the basic elements utilized in designing our model. The bidirectional arrows represent the relations between the *signatures* and the *universes* while provisioning resources during the multimode or simple auto-scaling. Figure 11 shows *universes* interacting with unidirectional and bidirectional arrows.

Step 2: Design and implement the model's *ASM Transition Rules* to reflect the job execution phases.

Step 3: Refine algorithms from the two categories of auto-scalers offered with DISSECT-CF, with *Transition Rules*.

Step 4: Evaluation of the model with the *Transition Rules* and evaluation goals.

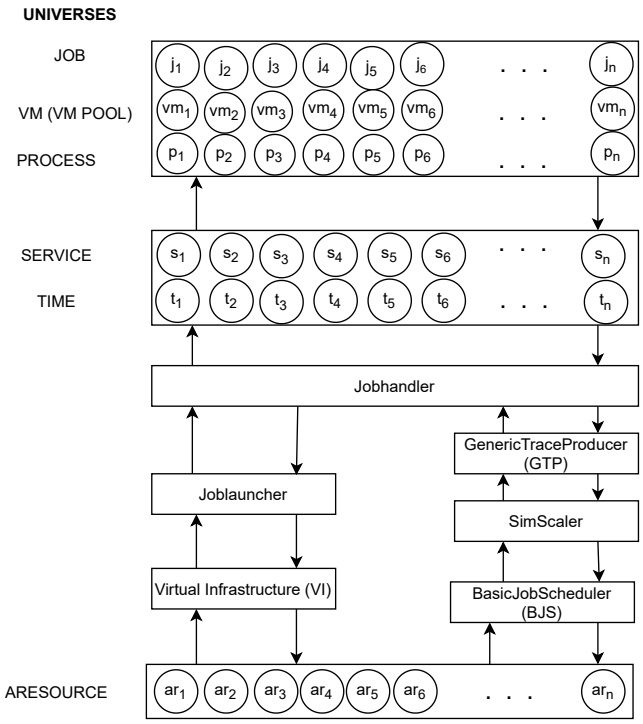


Figure 9: Basic elements of the ASM model for Auto-Scaling

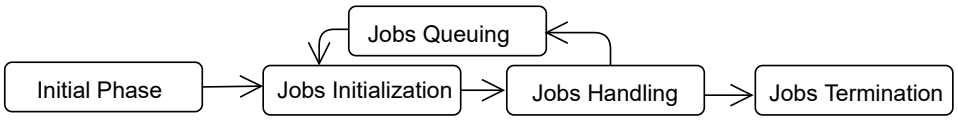


Figure 10: ASM Modelled Auto-Scaling Phases

Step 5: Model Validation with validation goals on test cases created from existing auto-scalers. Computational Tree Logic (CTL) formulae were applied for Astam's verification.

ASM functions (seen in table 3) were designed to relate to *Astam's universes* (as shown in figure 9).

5.3 Refinement of the Multimode and Simple Mechanisms

Our *Astam* model comprises of five *Transition Rules* as seen in figure 10. These rules are designed to reflect the job execution phases an auto-scaler undergoes during job processing. The rules enable users to analyse the VM provision behaviours of auto-scalers. We utilised algorithms to express the

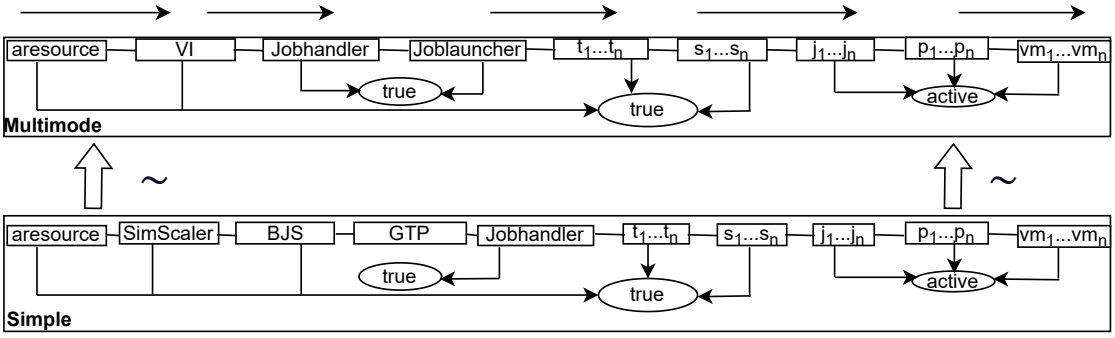


Figure 11: Ground Model for ASM Auto-Scaling Job Initialising

Algorithm 1 *Simple Jobs Initialising*

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**
  $processState(p) := ready$
end if
while $SimScaler(j, vm) \wedge BJS(j, vm) \wedge GTP(j)$ **do**
  if $jobRequest(j, ar) = true \wedge processRequest(p, ar) = true$ **then**
    $JobTime(j) := started$
  end if
  if $mappedVM(j, p) = true \wedge mappedJob(j, p) = true$ **then**
    $JobTime(j) := started$
  end if
  if $installed(j, vm) = true \wedge Jobhandler(j, vm) = true$ **then**
    $JobState(j) := submitted$
  end if
end while

details of our *ground model* shown in figure 11. These algorithms were further refined according to the ASM refinement method. Our ground model and the refinements are later compared for equivalence according to *Börger's refinement* to check for the consistency of state transitions.

The ASM Transition rules are: (i) Initial Phase (ii) Job Initialising (iii) Job Queuing (iv) Job Handling and (v) Job Termination. Derived functions were were introduced to ensure modularisation. In the next sub-sections, we discusss the job initialising phase to illustrate *Astam's* refinement process via transition rules. State transitions are highlighted to reflect the state changes during job processing. Now let us discuss Astam's Job Initialising Phase.

Astam's job initialising phase (figure 11) begins with a system call. Uni-

Table 3: List of ASM Functions

JobState: Job → {*idle*, *submitted*, *waiting*, *running*, $failed$, *done*}
ProcessState: Process → {*new*, *ready*, *waiting*, *running*, *stopped*}
SystemRequest: Request × AResource → { $true$, $false$}
SystemState: InfraState → {*idle*, *active*, *waiting*, *busy*, *stopped*, *done*}
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 → {$undef$, $true$, $false$}
InitReslist: $IReslist$→ { IRL_{active}, IRL_{idle}, IRL_{busy}}
InitReqFunctions: $InitReqFun$→ { IRF_{active}, IRF_{idle}, IRF_{busy}}
Job: Process → Job
Jobhandler: Job → Joblauncher, Job → VM
NumofSerReq: NumofSerReq→ { $undef$, Num_{min}, Num_{avg}, Num_{max}}
WorkloadPrediction: PredWorkload → { PWL_{active}, $PWL_{inactive}$}

verses are assigned to foster job processing. This transitions *processState* from *new* to *ready* as seen in lines 2 of our ground model algorithms 1 and 2. The auto-scalers specific universes are provisioned to monitor the activities of jobs and VM 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. VMs are mapped to 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 *joblauncher* in *Multimode* auto-scalers. The *JobTime* and *Jobstate* are updated to *started* and *submitted* as seen from lines 12. The refinement of algorithms 1 and 2 are modelled in algorithm 3.

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

Algorithm 2 *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**

in line 7. This refinement is equivalent to the algorithms 1 and 2 and the job initialising phase of our ground model as seen in figure 11. Since the state changes of algorithm 3 are equivalent to the state transitions of our the ground model after the ASM run.

The equivalence of state transitions achieved during the ASM runs via the application of derived functions helped us to conclude that, although the auto-scalers were developed on different frameworks, they exhibited similar VM provision behaviours during job processing. In the next section, we will evaluate *Astam* and its applicability.

Algorithm 3 *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 4 *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**

6 Astam Evaluation and Validation

Our *Astam* model was evaluated emphasizing the application of the ASM theory. Test Cases were generated from the formalized algorithms and validated on CoreASM toolkit to assess the application of *guarded updates*. Now let us discuss the application of Astam's job initialising phase on the *Vmopt* auto-scaler.

6.1 Astam Model Evaluation

Algorithm 4 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. *InitReqFunctions* is applied to provision *Aresources* 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. 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 4. This evaluation process is applicable to all auto-scalers. This refinement is equivalent to the job initialising of our model shown in figure 11 and algorithm 3, since the derived functions can be refined back to the ground model.

6.2 Adoption of Astam with Other Auto-Scaling Algorithms

The related works analysed past auto-scalers mechanisms, and selected [10] 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. [10] presents an approach using workload prediction, as well as horizontal and vertical scaling. Therefore, it was possible to analyse and classified it as a *multimode* auto-scaler due to its specific features. Figure 12 shows the

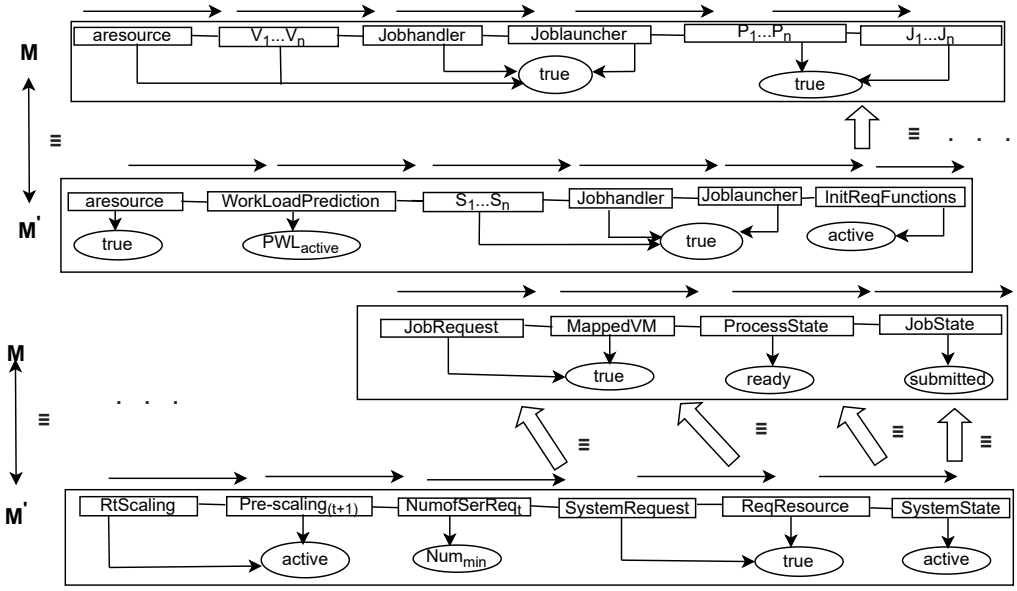


Figure 12: Workload Predicted Auto-Scaler Job Initialising

refining process of our ground model (M) to [10]'s refinement(M') at the job initialising phase. Derived functions (*InitReqFunctions*) are employed to foster the resource provisioning. This causes the prediction of Workloads at the commencement of job processing. Resource scaling up and pre-scaling $(t+1)^{th}$ Interval are activated per user service requests. *SystemState* transitions to *active*. Figure 12 shows the equivalence between M and M' due to the application of derived function; which can be refined back to the ground model in accordance to the ASM method.

7 Conclusion and Future Research Direction

This dissertation focused on the analysing MR simulators and modeling resource provisions to meet user demands. As such, three research efforts were carried out. First, a *MaReClass* framework was designed and implemented to simplify the selection of MR simulators for research and development. Second,a *Haspeck* solution was designed and implemented to improve job performance on MRH. Third, an *Astam* model was designed for analysing the VM provision behaviours of cloud auto-scaling mechanism.

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.

Our research efforts in our dissertation has fostered the development of three major contributions to science. Now, let us outline these contributions.

7.1 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 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.

References

[1] 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.

[2] 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.

[3] 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.

[4] Mandana Farhang and Faramarz Safi-Esfahani. Recognizing mapreduce straggler tasks in big data infrastructures using artificial neural networks. *JOURNAL OF GRID COMPUTING*, 2020.

[5] Khushboo Kalia and Neeraj Gupta. Analysis of hadoop mapreduce scheduling in heterogeneous environment. *Ain Shams Engineering Journal*, 12(1):1101–1110, 2021.

[6] Gabor Kecskemeti. Dissect-cf: a simulator to foster energy-aware scheduling in infrastructure clouds. *Simulation Modelling Practice and Theory*, 58:188–218, 2015.

[7] Najme Mansouri, R Ghafari, and B Mohammad Hasani Zade. Cloud computing simulators: A comprehensive review. *Simulation Modelling Practice and Theory*, 104:102144, 2020.

[8] 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.

[9] 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.

[10] 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.