Thesis for the Degree of Masters

Interference-Aware Resource Scheduling Using Application Clustering Analysis In Virtualized Environments

가상 환경에서의 응용 군집 분석을 이용한 간섭 고려 자원 스케줄링 기법

By

Adufu Theodora Kafui Adzo

Department of Computer Science

The Graduate School

Sookmyung Women's University

Thesis for the Degree of Masters

Interference-Aware Resource Scheduling Using Application Clustering Analysis In Virtualized Environments

가상 환경에서의 응용 군집 분석을 이용한 간섭 고려 자원 스케줄링 기법

By

Adufu Theodora Kafui Adzo

Department of Computer Science

The Graduate School

Sookmyung Women's University

Interference-Aware Resource Scheduling Using Application Clustering Analysis In Virtualized Environments

By

Adufu Theodora Kafui Adzo

A Thesis submitted to the

Department of Computer Science and

the Graduate School of

Sookmyung Women's University

in partial fulfillment of the requirements

for the degree of Master

In charge of major work: PROFESSOR KIM YOONHEE

June 2016

Interference-Aware Resource Scheduling Using Application Clustering Analysis in Virtualized Environments

This certifies that the degree of Master of

Adufu Theodora Kafui Adzo is approved by

____ (Signature)

Chair of Committee

____ (Signature)

Committee Member

____ (Signature)

Committee Member

The Graduate School

Sookmyung Women's University

June 2016

ACKNOWLEDGEMENT

First and foremost, I would like to thank the Almighty God for the strength, ability and unparalleled wisdom He made available to me throughout this research. I also thank His able Clergy who have always encouraged me to challenge myself and strive for excellence.

I have been very fortunate to have a superb advisor, Professor Yoonhee Kim whose professional abilities and knowledge are always admired. She provided me with excellent guidance, allowed me to investigate my own ideas, gave me many opportunities to learn, and helped me conduct research which has led to the success of this thesis. When I encountered obstacles in research, discussions with her often inspired me. Her endless amount of energy is really an inspiration for me to do better research and I greatly thank her for all the kindness, support and encouragement.

Many thanks to my thesis reviewers and the defense jury committee, Professor Byeong Mo Chang and Professor JongWon Choe, who have all given me excellent guidance and advice. I am grateful for the tremendous amount of time and effort that they have contributed to this work.

I am blessed to have a knowledgeable faculty, supportive colleagues, loving families and friends. Here in Korea, my research colleagues in the Distributed and Cloud Computing Lab at Sookmyung University, the Nam family, Salem Worship team, members of Winners' Chapel International Seoul and all my faithful friends, provided a great deal of support in putting together this thesis. I cherish the times we spent together discussing issues, working and dinning. And in Ghana, the faculty of the Computer Science Department, headed by Dr. Ferdinand A. Katsriku, have in no small measure spurred me on to higher heights. Thank you.

Finally, I appreciate my parents Mr. and Mrs. Adufu, and my best friend Eric Egan, who passed on to me a strong work ethic and the mentality of leaving a lasting legacy in addition to their my ever-present support. Together, your strong convictions about my bright future is admirable and worthy of emulation. I am forever grateful for your unflinching support, encouragement, and assistance during the writing of this thesis.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABSTRACT

Virtual technologies such as container technologies and cloud platforms provide scalable on-demand resource provisioning for the execution of scientific applications and thus are being deployed more and more in research and in production environments. However, without an efficient performance isolation layer, it is not guaranteed that applications sharing similar resources would not impact upon each other. In this thesis, an interference-aware scheduling method that mitigates the problem of performance interference based on the results from clustering analysis of application characteristics, is proposed. The proposed method detects the amount of interference between the applications waiting to be scheduled in OS-level virtualization environments and co-schedules them in a manner that reduces contention for shared resources. The method is compared with other container cluster scheduling techniques and results show that the proposed method reduces performance interference and hence improves the performance of applications.

Keywords: OS-level virtualization, Container, Application characterization, Interference-aware, k-means, Interference

I. INTRODUCTION

1. Research Background and Problem Definition

Virtualization provides efficient use of computing resources by enabling dynamic resource provisioning during application executions. Particularly, cloud computing provides scalable on-demand resource provisioning for the execution of scientific applications. Recently, lightweight containers such as Linux containers are also gaining significant attention as efficient virtual technology solutions. Due to their almost zero start-up times, minimal run-time overhead, and the relatively higher deployment density per physical host [1, 2, 3] researchers are deploying containers to facilitate reproducibility of scientific workflows [4], enhance security systems [5, 6] benchmark virtual machines [7] and provide distributed storage [8]. As a result, it is obvious that in coming years, container technologies will be deployed in more areas of research and in production environments. Consequently, allocating resources to the containers to ensure peak performance in any research environment would be of utmost importance.

However, without an efficient performance isolation layer, it is not guaranteed that applications sharing similar resources would not impact upon each other. Different Container Cluster Managers (CCM) manage the placement of containerized applications across multiple hosts each with a different approach. Most strategies tend to maximize resource utility by placing most containers on a node to increase the deployment density per physical host of containers [3]. In idealized environments, the co-located containers should not be affected by other applications however, due to the resource sharing by containers on a physical node, there is bound to be contention for shared resources and hence performance interference by co-executed applications. In fact, interference among containers is proportional to the competition among co-located applications for shared resources. According to [9,10,11], the execution time of a task running on a host machine with other tasks, relative to the execution time on a host machine alone, increases in direct proportion to an increase in the number of processes co-running on the host machine. Knowing the interference caused by different applications concurrently running on a given node is therefore of utmost importance during effective scheduling of containers in virtual environments using current scheduling methods [12, 13, 14].

2. Research Significance and Approach

In this thesis, an interference-aware scheduling method that mitigates the problem of performance interference based on the results from clustering analysis of application characteristics, is proposed. The main design question for the proposed scheduler is how to model an efficient scheduler that schedules containers to resources based on the characteristics of the applications running on them in order to reduce interference from co-located applications. First, each application will have different resource needs, based on its computation, memory usage and data access indicating that the level of interference will vary per application depending on the characteristics of the co-executed applications. Second, the scheduling system must maximize performance at the lowest possible interference.

This thesis proposes an interference-aware scheduler that considers the type of the application and the resource usage (CPU, Memory) of that application according to results from clustering analysis. In order to improve the scheduling decisions of the scheduler, first, an interference ratio is calculated according to a proposed equation. With the results, selection of co-scheduled applications and resources with is done according to the least expected interference by the co-executed application. Nadeem et al. [9], Zhang et al. [10] and Chiang et al. [11] through their respective experiments ascertain that, the execution time of a task running on a host machine with other tasks, relative to the execution time on a host machine alone, increases in direct proportion to an increase in the number of processes co-running on the host machine. Following the above considerations, this thesis considers interference as the change in the total execution time of an application. In this thesis, the proposed scheduler aims to minimize performance interference of containerized applications and thus improve the throughput of applications. The thesis also considers a scheduling environment which enables the user to indicate Service Level Agreements (SLAs) by which executions should be made. In line with this, the proposed method makes the following contributions:-

- Clusters a set of applications based on their CPU and memory usage characteristics using K-means clustering method.
- Calculates an interference ratio for a new application based on the Euclidean distances between that applications and other applications in a K-means cluster.
- Proposes an interference-aware scheduling method which co-schedules tasks to minimize interference based on an interference ratio.
- Schedules applications in a manner that maximizes the number of applications that are executed from a queue at stable performance.

3. Thesis Structure

This thesis is structured into 5 chapters as follows: Chapter I describes the research background with details of the research problem, the significance of the research and the structure of the thesis. Chapter II presents related works in four categories: (1) Performance evaluation of OS level virtual environments, (2) Application characterization methods, (3) Interference-aware scheduling techniques and (4) Scheduling in Container Cluster Managers (CCM). Chapter III describes the proposed Interference-aware Scheduling technique with descriptions of the profiling process, the K-means clustering method[15], the proposed interference-aware detection method and associated scheduling algorithm. In Chapter IV, the proposed method is evaluated through various experiments and the thesis is concluded in Chapter V.

II. RELATED WORKS

Operating System-Level Virtualization such as FreeBSD Jails[16], Linux VServer[17], Solaris Containers[18], Softricity[19], Virtuozzo [20] and the recently popular Docker Containers[21], are widely being deployed for research and production purposes. Containers allow for multiple isolated user-space abstractions for guest processes without the need to host a Guest OS, thus are more lightweight than traditional virtualization technologies such as Virtual Machines (VM). In this section, some related works under four major categories are described: (1) Performance evaluation of OS level virtual environments, (2) Application characterization methods, (3) Interference-aware scheduling techniques and (4) Scheduling in Container Cluster Managers (CCM) as follows:

1. Performance Evaluation of OS-Level Virtualization Technologies

Resource isolation and security isolation are examined for some container-based systems using benchmark experiments by Stephen et al[22]. Their experiments demonstrate that container-based systems are suitable for usage scenarios that require high levels of isolation and efficiency such as server-type workloads. From their experiments, container systems perform two times better for server-type workloads than hypervisor-based systems. Miguel et al[23, 24] however, evaluate the performance of container-based virtualization for HPC environments. In their papers, benchmark experiments were conducted to show that an isolation layer with least overhead is essential for better resource sharing in user-oriented custom HPC environments. From the results of both experiments, container-based systems performed better than HPV systems represented by Xen, in processing, memory, disks and network tests. Xen however had better isolation due to non-shared Operating System (OS). Their experiments indicate that the performance of HPC applications in container-based virtualization systems could experience interference when applications requiring similar resources are co-scheduled due to relatively poor resource isolation. This also indicates that for HPC scientific environments the absence of performance interference cannot be guaranteed for all types of applications.

According to the experiments in Honeynet[25] the proposed system is able to isolate compromised honeynets from other honeypots on the server with container-based technology thereby increasing security and system stability. Deployment and maintenance of “honeypots” is hence faster and easier for low physical resource utilization using LXC container-based virtualization. However, the isolation of "honeynets" described in their research is focused on isolating networks which have been compromised due to attacks by hackers, and not performance isolation.

2. Interference-Aware Scheduling

TRACON[11] uses modeling and control techniques on application statistics, VM statistics and interference profiles, to predict interference in para-virtualized environments. With the results of their prediction, incoming tasks are co-scheduled to resources with applications of least interference using a Minimum Interference Batch Scheduler (MIBS). MIBS is able to improve performance however, TRACON's proposed method is limited to only para-virtualized environments since it measures interference by the amount of I/O operations performed by the virtual device driver and the native driver of para-virtualized systems. The methods were also evaluated using simulations and thus cannot be validated in real environments. On the other hand, the proposed method considers different parameters of applications in a relatively cost-effective manner.

DejaVu[26] uses an "interference index" to estimate interference between workloads. With the information from the index, DejaVu profiler is able to increase the accuracy of estimated required resources in order to maximize performance. Determining the interference index requires profiling of low-level metrics of workloads over many hours, however, workload clustering analysis is only used to determine which sets of applications require interference-aware scheduling and not the amount of interference introduced by co-scheduling of applications. The proposed method however, considers interference for all applications in the service system.

3. Scheduling by Container Cluster Managers, (CCM)

Container cluster managers orchestrate multiple containerized applications across multiple hosts. Each of these CCMs have scheduling policies and strategies which manage the provisioning of resources to various containers for the execution of different types of applications deployed on them.

Google's container-based cluster manager, Kubernetes[27], uses an automatic bin-packing scheduling scheme to place containers into pods based on their resource requirements and constraints. Scheduling is done using two user-configurable run-time scheduling policies: FitPredicate[12] and the PriorityFunction[13]. The scheduler ranks the machines that meet all of the rules indicated in the FitPredicates, and then chooses the best one according to the PriorityFunction. However, these policies do not consider interference from running applications during the placement of containers.

Similarly, Apache Mesos[28], a distributed systems kernel that uses a two-level scheduling scheme to manage task scheduling, resource allocations and sharing in a heterogeneous cluster environment, offers resources to Frameworks such as Apache Marathon[14] using the Dominant Resource Fairness Algorithm. The Frameworks schedule an application to an executor process or a container launched on slave nodes. Tasks are however not scheduled to resources with consideration of the effects of interference on the performance of the co-scheduled applications.

Docker Swarm[29], the native clustering tool for Docker containers employs a manager/agent deployment structure which includes a host that runs a Swarm manager and other hosts which run a Swarm agent each. The Swarm manager orchestrates and schedules containers on the hosts whilst the Swarm agents enable the execution of applications in Docker containers. Docker Swarm easily provisions and manages large numbers of containers in node clusters during large-scale HPC scientific experiments according to 3 scheduling policies: bin-packing, spread and random[30].

The bin-packing strategy chooses physical resources based on the highest number of containers running on that resource whilst the random scheduler does not consider the number of running containers. The spread strategy on the other hand, schedules applications to physical resources with the least number of containers running on them and serves as the container placement strategy adopted by the proposed method. The proposed method however, considers the interference between applications based on results from clustering analysis in order to mitigate interference.

4. Comparison of Related Works

From Table 1, it is observed that current Container Cluster Managers schedule containers to physical resources without considering performance interference. Scheduling methods which consider interference are also deployed in hypervisor-based virtualized platforms (VMs) and para-virtualized platforms and may not be applicable to OS-level virtualized environments (Containers) without major modifications. The proposed method introduces the concept of interference-awareness into container environments based on analysis from application characterization.

TABLE 1. COMPARISON OF RELATED WORKS

RELATED WORKS	JOB TYPE	CHARATERISTICS OF SCHEDULING METHODS				
		Application characteristics	Application clustering analysis	Resource scheduler	Interference-awareness scheduling	Platform
WORK 1 [27]	Variable	△	╳	Mesos /Marathon	╳	Containers
WORK 2 [26]	Variable	△	╳	Kubernetes	╳	Pods/ containers
WORK 3 [29]	Variable	△	╳	Docker Swarm	╳	Containers
WORK 3 [14]	Data Intensive	○	╳	MIBS	○	Virtual Machines (VM)
WORK 4 [25]	Workload	○	○	DejaVu	○	Virtual Machines (VM)
PROPOSED WORK	Variable	○	○	Interference-aware Scheduler	○	Containers

III. INTERFERENCE-AWARE SCHEDULING BASED ON APPLICATION CLUSTERING

In this thesis, the proposed interference-aware technique illustrates a scheduling system of multiple applications waiting to be scheduled unto container resources. It starts when a new application joins the queue with profile information on some application characteristics. First, the new application together with all the tasks in the queue are re-clustered into specified number of clusters in a manner that minimizes the Within the Cluster Sum of Squares (WCSS), using the k-means algorithm. Next the distance between each application and the centroid of the cluster in which the new application is found is detected and the applications are scheduled to resources according to the interference ratio.

Thus, the proposed scheduling technique begins with ***Application Clustering (AC)*** of all waiting applications using K-Means algorithm and performs ***Interference Detection (ID)*** between the clustered applications, as part of our ***Interference-Aware Scheduling Algorithm (IAS).*** It then invokes a ***Resource Selection Method (RSM)*** to place applications in containers on physical resources. The proposed interference aware technique is illustrated in Figure 1.

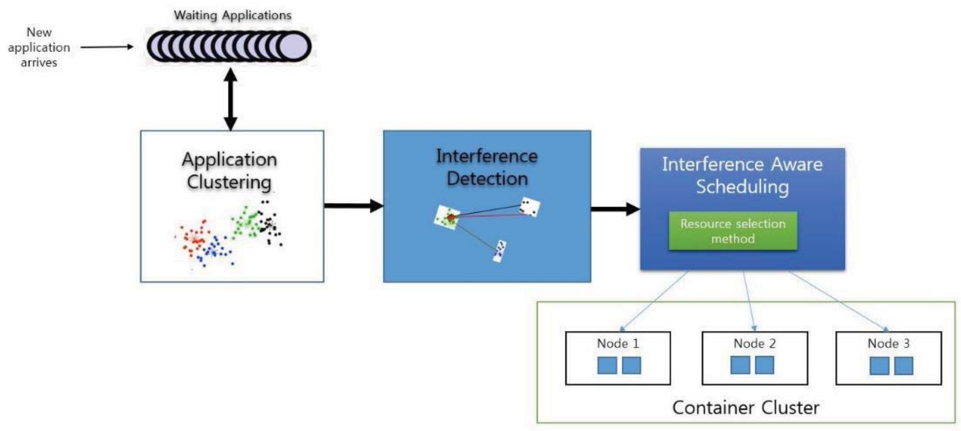

FIGURE 1: INTERFERENCE-AWARE SCHEDULING BASED ON APPLICATION CLUSTERING

TABLE 2: NOTATIONS AND DESCRIPTIONS

Notation	Description
$\mathbb{Q}$	a queue with a set of q_i applications
q_i	an i^{th} application in a queue $\mathbb{Q}$ with resource usage profile
Cluster(i)(x_h, c_i)	An i^{th} Cluster containing x_h data-points for h = {0,1,$\cdots$, n }, and centroid c_i
x_h	An h^{th} data-point in Cluster(i)
x_j	An h^{th} data-point in Cluster(i) selected for co-execution
c_i	the centroid of Cluster (i)
r_{jh}	The interference ratio between x_j and x_h
α	the threshold value for interference
CalcDist(x, c)	Calculating the distance between data-point and centroid
CalcR(x, c)	Calculating interference between two points
Sort(x, c)	Sorting the data-points based on the interference ratio
policy	The policies indicated by the user (default: no-schedule)
$\mathbb{N}$	a list of nodes in the node cluster
n_i	An i^{th} node from the list of nodes in the node cluster
$cont_h$	An h^{th} container on node(i)

Each of the components is described in the following sub-sections and the key notations used are listed in Table 2.

1. Application Clustering Using K-Means

In recent computing research, unsupervised machine learning techniques such as cluster analysis are used to provide solutions to the NP-hard problem of application characterization since there is no definite way of grouping data elements.

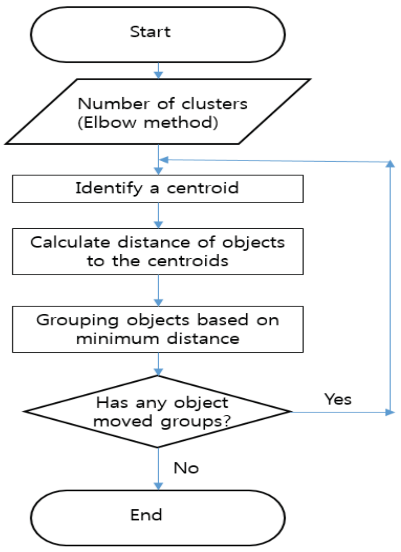
Start
Number of clusters (Elbow method)
Identify a centroid
Calculate distance of objects to the centroids
Grouping objects based on minimum distance
Has any object moved groups?
Yes
No
End

FIGURE 2: FLOWCHART OF THE K-MEANS ALGORITHM [15]

In this thesis, the K-means[15] clustering algorithm as shown in Figure 2 is deployed as the method of characterizing applications because it is fast, robust and relatively efficient. This method effectively partitions data into Voronoi cells[31]. The outcome of the partitioning contains multiple sets each containing a unique center or centroid and the Euclidean distances of the samples in a set to the set's center must be smaller than to any other sets' centers.

For a given set of data points, X ={x_1, x_2, x_3,...,x_n} with initial centroids given as C = {c_1, c_2, ..., c_m}, the algorithm calculates the distance between each data point and cluster centroids. It then clusters the data points into **k** clusters and computes the new cluster centroid. If no data point was reassigned then the algorithm ends, otherwise it classifies the data points and recalculates the new cluster centroid in an iterative process.

However, the algorithm is naive in that, it trades off quality for compactness of representation. That is, it does not consider the accuracy of the value of k but only classifies the data points into k number of clusters. Given various numbers of sets, k, the clustering results may vary largely hence the need to determine the right number of clusters. Among the various methods of determining k such as *gap statistics*[32], *Information criterion approach*[33] and *cross-validation*[34], the *elbow method*[35] is used to determine the number of clusters, **k**. This method looks at the percentage of variance explained as a function of the number of clusters[35].

To calculate **k**, using the elbow method, first calculate the sum of the squared distance between each member and its centroid, for different values of **k.** The calculation, known as finding the Sum of Squared Error (SSE) is done with equation 1.

$$SSE = \sum_{i=1}^{K} \sum_{x \in c_i} dist(x, c_i)^2 \qquad (1)$$

With the results for various values of k, the k at which SSE changes abruptly is selected as the value of k. At this point, the grouping of the applications is done into k number of clusters.

The K-means clustering algorithm aims to group the available applications into **k** sets based on the application's CPU usage and memory usage characteristics. The clustering is such that, the Within Cluster Sum of Squares (WCSS) between the data-points is minimized for each cluster according to equation 2.

$$wcss = \sum_{J=1}^{K} \sum_{i=1}^{n} ||x_i^j - c_j||^2 \qquad (2)$$

2. Interference Detection

The proposed interference detection service is based on the concept of using distances between clusters to estimate workloads and their resource utilization[36] coupled with the concept of estimating interference between cells in cellular clusters using the distances between cells in cellular networks[37]. In this thesis, interference detection is done through the computation of an interference ratio between a target application, $x_k(a, b)$ and all the applications in the application cluster such as $x_j(a, b)$ in Figure 3.

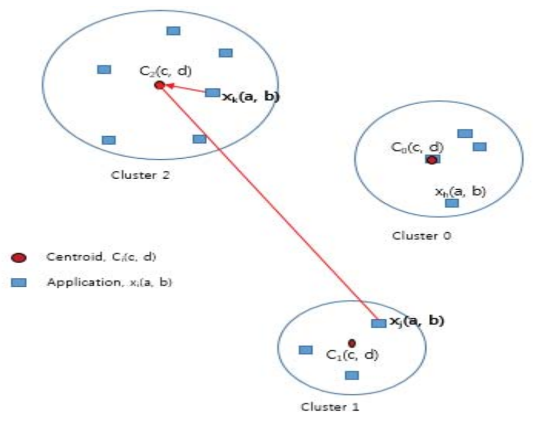

FIGURE 3: INTERFERENCE RATIO CALCULATION USING DISTANCES

First, a two-dimensional Euclidean distance between a target application, x_k(a, b) and the centroid of their cluster, C_2(c, d) and that of other applications, x_j(a, b) and the centroid of the target application, C_2(c, d), is computed and noted according to equation 3. Next an interference ratio is calculated according to equation 4.

$$\mathrm{dist}(x_k, C_2) = \sqrt{((a-c)^2 + (b-d)^2)} \qquad (3)$$

$$\text{Interference ratio, } r = \frac{\mathrm{dist}(xk,\ C2)}{\mathrm{dist}(xj,\ C2)} \qquad (4)$$

3. Interference-Aware Scheduling

The proposed interference-aware scheduling technique is based on batch scheduling[11]. It however includes the clustering of applications based on their resource usage in containers to show a scheduling procedure for multiple applications waiting to be scheduled in a manner that mitigates interference. The Interference-Aware Scheduling (IAS) algorithm, Algorithm 1, considers the characteristics of an application and the interference between that application and co-scheduled applications before assigning the application to resources.

Scheduling of applications begins when a new application enters the queue, Q, of waiting applications = {q_i for i=1, 2, 3⋯ n}. All the applications in the queue are characterized into k clusters as mentioned in previous sections. The value of k is determined using the elbow method. It starts when a new application joins the queue with profile information on CPU utilization and memory usage.

Algorithm 1 Interference-aware Scheduling Algorithm

```
Input: A Queue $\mathbb{Q} = \{q_i \mid i = 1, 2, \cdots, n\}$ , SLA {deadline, priority, etc}
 1: New application arrives;
 2: while ($\mathbb{Q}$ =! 0) do
 3:     k ← FindK(Elbow);
 4:     Cluster(i)($x_h$, $c_i$) ← Cluster(K-means);
 5:     $x_j$ ← Select$X_j$($x_h$, $c_i$);
        dist($x_j$, $c_j$) ← CalcDist($x_j$, $c_j$);
 6:     for each $x_h$ in Cluster(i)($x_h$, $c_i$) do
 7:         dist($x_h$, $c_j$) ← Calcdist($x_h$, $c_j$);
 8:         $r_{jh}$ ← CalcR(dist($x_j$, $c_j$), dist($x_h$, $c_j$));
 9:     end for
10: end while
11: $x_h$Schedule ← Sort($x_h$, $r_{jh}$);
12: $x_h$ ← SelectMin($r_{jh}$);
13: if $r_{jh} \leq \alpha$ then
14:     Perform resource selection;
15: else
16:     if $policy == no-schedule$ then
17:         Fail($x_j$);
18:     end if
19: end if
Output: Co-execute $x_j$, $x_h$
```

First, the new application together with all the tasks in the queue is re-clustered into specified number of clusters in a manner that minimizes the Within the Cluster Sum of Squares(Lines 2), using the k-means algorithm. A target application is randomly selected from the queue and the interference between the target application and other applications in the service system is detected using the proposed interference ratio(Lines 5-9). After, applications are sorted in order of increasing levels of interference(Lines 11). The scheduler then selects a co-application with the least interference ratio(Lines 12) and compares the interference ratio to a preset threshold value. If the ratio is less than the threshold value, the scheduler then invokes the resource selection method to select a suitable node from the cluster(Lines 13-14). The method returns nodes for the execution of each pair of applications into containers according to their resource demands. The applications are co-scheduled to containers on the node. Otherwise, the service system considers the policies indicated by the user in order to invoke the resource selection method. In this service system, the policy used is the no-schedule policy. The no-schedule policy requires that the selected application is not scheduled when the minimum interference ratio is higher than the threshold.

4. Resource Selection Method

Interference-aware scheduling methods for scientific applications aim at minimizing the performance over-heads introduced by co-scheduling applications on a particular physical resource. It is therefore important that in a container cluster, the resource that minimizes the likelihood of interference is selected. This thesis adopts a Scheduling Strategy that prefers nodes with the least number of containers with the assumption that the number of containers is directly proportional to the amount of interference[30]. The method is illustrated in algorithm 2.

Algorithm 2 Resource Selection Method

```
Input: NodeList $\mathbb{N} = \{(n_i) \mid i = 0,1,\cdots, m\}$, Co-scheduled applications
    $[x_h(min_C, min_R)]$, $[x_j(min_C, min_R)]$
 1: Set resource available == true;
 2: Node, $n_i \leftarrow FindNode(min_{cont})$;
 3: if resource available == true then
 4:     $cont_h \leftarrow Create[\mathrm{x}_h(min_C, min_R)]$;
 5: else
 6:     Stopped($cont_h$) ←FindStopped($n_i$);
 7:     if Stopped($cont_h$) > 3 then
 8:         for each $x_h$ in co-scheduled applications do
 9:             $cont_h$ ←Restartcont[$\mathrm{x}_h(min_C, min_R)$];
10:         end for
11:     end if
12:     Results ← $Execute(\mathrm{x}_h)$;
13: end if
Output: Execution Results
```

When the resource selection method is invoked, the scheduler looks for available nodes with the least number of containers based on the assumption that, the interference for resources increases with the number of containers on each physical resource (Lines 1). The method proceeds to find stopped containers on the node for restarting, otherwise, the method creates containers based on the required memory and CPU resources of the applications. When there are sufficient number of stopped containers, the method restarts two stopped containers according to the required resources (Lines 2-7). Otherwise, the method creates new containers into which the resources would be scheduled (Lines 9-10). The method returns the results of executing the applications in the containers to the scheduling algorithm.

Ⅳ.EXPERIMENTS

Other experiments that validated the proposed interference-aware technique are presented in this section. The experiment environment, target applications, and the experimental setting along with experimental results is described in the following sections.

1. Experiment Environment

Docker containers[21], a lightweight virtualization solution for fast creation of containers and execution of applications independent of hypervisor layer, is deployed as the execution infrastructure for all the experiments. To manage the scheduling of Docker containers, Docker Swarm[29] is deployed. This container cluster consists of two nodes on the same local network with one node serving as both the Docker Swarm Manager and a Swarm Agent whilst the other node serves as a Swarm Agent only. The nodes in the proposed service system have a total RAM of 8GB and 12 CPU cores for the Swarm Manager, and total RAM of 8GB and 4 CPU cores for the Swarm Agent node respectively. Each of these server machines are operated by Ubuntu Trusty Tahr Operating System. To facilitate accuracy of the results, Ubuntu 14.04-based container images which have been pre-installed with the scientific applications are used for these experiments.

① Scientific Applications

Scientific applications are classified into bag of tasks (BOT) and workflows[38] represented as Direct Acyclic Graphs (DAG). In the experiments, a selection of 4 scientific applications, each with different CPU and Memory usage characteristics is made and their resource usage data for different parameters is profiled as shown in Table 3. The experiments include 3 BOT applications, CFD[39], Melt[40], Peptide[41] and 1 workflow application, Montage GALFA[42] as explained below.

A) Montage GALFA

Montage[42] is a high performance Astronomical Image Mosaic Engine for creating composite FITS (Flexible Image Transport System) mosaics using multiple astronomical images. Due to the large input, intermediary and output data sizes, this application is considered a data-intensive application.

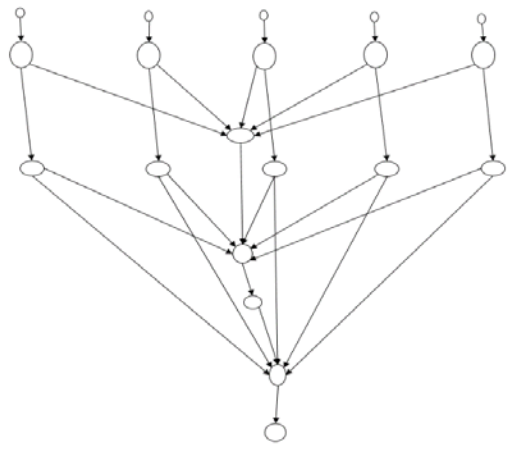

FIGURE 4: MONTAGE GALFA WORKFLOW APPLICATION

In this thesis, a set of five data cubes is obtained from the Galactic

Arecibo L-band Feed Array HI survey[43] and the data cubes are aggregated into a mosaic following three major steps as shown in Figure 4. First, the input data cubes are shrunk by averaging different number of planes (5, 10, 15, 20, 25 in this thesis) in the wavelength axis. Then, the shrunk images are re-projected to map the input pixel space to sky coordinates and to the output pixel space. Finally, the re-projected images are aggregated to produce the final mosaic.

B) Computational Fluid Dynamics (CFD)

The second target application is a CPU-intensive application known as Computational Fluid Dynamics simulation application which is used for 2-dimensional Euler unsteady flow analysis.

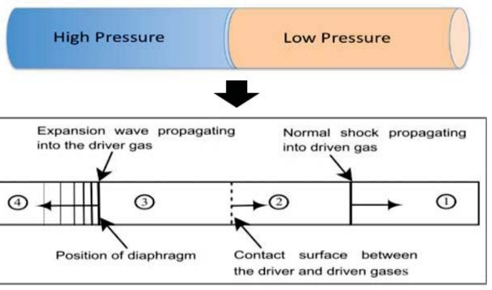
High Pressure
Low Pressure
Expansion wave propagating into the driver gas
Normal shock propagating into driven gas
④
③
②
①
Position of diaphragm
Contact surface between the driver and driven gases

FIGURE 5: THE SHOCK TUBE PROBLEM IN CFD

The application example used in this experiment is the 2-D shock tube problem using aerodynamics grids in Figure 5. In this experiment, grid mesh of three different sizes, 2KB, 32KB, 512KB, are used as input data. Each size of grid mesh data translates to a different memory and CPU usage profile.

C) Large-scale Atomic/Molecular Massively Parallel Simulator

(LAMMPS)

The third is a molecular dynamics code to model an ensemble of particles in a liquid, solid, or gaseous state. It can model atomic, polymeric, biological, metallic, granular, and coarse-grained systems using a variety of force fields and boundary conditions. The simulator is used for simulating different types of particle behaviors. In this experiment, Melt and Peptide problems are solved using their corresponding application ensembles.

a) Melt: simulates the rapid melting of a 3d LJ system.

b) Peptide: Peptide simulates the granular particle pour and flow of both 2d and 3d systems.

The above two applications are considered as separate applications in these experiments due to their different input data sets and execution modules. Also, due to their relatively stable memory usage across different runs, both Melt and Peptide applications are considered as memory intensive jobs.

② Application Profile Data

For the experiments, memory resource usage data and CPU utilization data is obtained for 4 different scientific applications with varying parameters for each application as shown in Table 3. The memory usage, CPU utilization information and the number of instructions executed by each application is obtained using Valgrind profiling tool[44] and htop system profiler[45]. In addition to memory usage data, Valgrind profiling tool is able to provide information on the number of instructions executed, the amount of cache misses, the call function as well as thread and memory errors.

TABLE 3: APPLICATIONS' CPU AND MEMORY USAGE PROFILES

Application	Memory usage, B	CPU utilization, %
GALFA (5 planes)	14,708,736	132.9
GALFA (10 planes)	14,716,928	126.2
GALFA (15 planes)	14,745,600	121.01
GALFA (20 planes)	14,766,080	119.5
GALFA (25 planes)	14,782,464	117.9
CFD (2KB mesh)	13,967,360	201.4
CFD (32KB mesh)	14,098,432	205.2
CFD (512KB mesh)	403,230,720	206.7
Melt (500 runs)	23,183,360	100
Melt (1000 runs)	23,314,432	100
Melt (1500 runs)	23,445,504	100
Melt (2000 runs)	23,576,576	100
Melt (2500 runs)	23,707,648	100
Melt (3000 runs)	23,838,720	100
Melt (3500 runs)	23,969,792	100
Peptide (500 runs)	32,460,800	99.9
Peptide (1000 runs)	33,382,400	99.9
Peptide (1500 runs)	34,213,888	100
Peptide (2000 runs)	35,606,528	100
Peptide (2500 runs)	36,810,752	100
Peptide (3000 runs)	37,613,568	100
Peptide (3500 runs)	38,817,792	100

Htop also, provides corresponding information on CPU and memory usage. In this thesis, only two characteristics, CPU and Memory usage, would be considered. For each characteristic, the data represented below is based on the maximum resource usage of each application during their execution life-cycle. From Table 3, it is observed that, the resource usage of applications is similar for changing parameters of that application.

2. Application Characterization

At the start of the experiments, **Application Clustering** is used to characterize the applications based on the application characteristic profiles (CPU and memory usage) of each application as shown in Chapter III. To find the number of clusters, k, which is most suitable for this experiment environment, the Simple K-Means method in Weka [46] is used to find the Sum of Squared Errors for different values of K. The results are presented in Figure 6. From the results of running the algorithm on the data set for various values of k as shown in Figure 6, the number of clusters which is most suitable for the experiment is 3. In the rest of the experiment, the number of clusters, k, is set to 3.

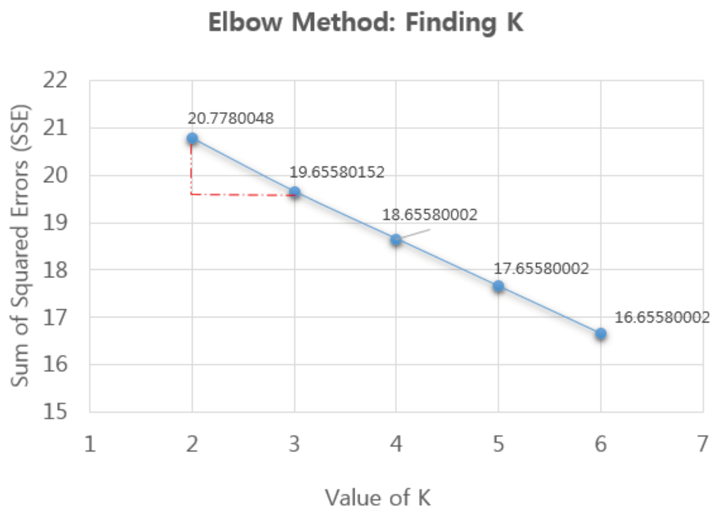

FIGURE 6: FINDING THE VALUE OF K USING THE ELBOW METHOD

Table 4 presents the details of the clustering analysis based on the

CPU usage and memory usage of 4 scientific applications when executed with different parameters, using K-means clustering technique.

TABLE 4: APPLICATION CLUSTERS AND INTERFERENCE RATIOS

Cluster	Application	Interference ratio, r
Cluster 0	CFD (512KB mesh)	0.00043
Cluster 1	Melt (500 runs)	0.01940
	Melt (1000 runs)	0.01910
	Melt (1500 runs)	0.01886
	Melt (2000 runs)	0.01858
	Melt (2500 runs)	0.01831
	Melt (3000 runs)	0.0186
	Melt (3500 runs)	0.01781
	Peptide (500 runs)	0.0093
	Peptide (1000 runs)	0.0089
	Peptide (1500 runs)	0.0085
	Peptide (2000 runs)	0.00797
	Peptide (2500 runs)	0.00754
	Peptide (3000 runs)	0.00727
	Peptide (3500 runs)	0.00691
Cluster 2	GALFA (5 planes)	1.0000
	GALFA (10 planes)	0.9535
	GALFA (15 planes)	0.8199
	GALFA (20 planes)	0.7450
	GALFA (25 planes)	0.6949
	CFD (2KB mesh)	0.2928
	CFD (32KB mesh)	0.3796

From the analysis, the sets of applications are grouped into 3 clusters based on their distances from the centroids of the clusters. The centroids of the Clusters are first initialized at (32460800, 99.9), (23838720, 100.0) and (23183360, 100.0) and finalized at (403230720, 206.7), (295672680, 99.9) and (14540800, 145.0) for Clusters 0, 1, 2 respectively through an iterative process of grouping applications in a way that minimizes the value of the Within Cluster Sum of Squares (WCSS). The distance between the applications and the finalized centroids is then calculated for the grouping of the applications into three clusters, Cluster 0, Cluster 1 and Cluster 2 respectively.

Cluster 1 has 15 applications, Cluster 2 has 7 applications whilst Cluster 0 has only 1 application, CFD (512KB mesh) according to the variations in the resource usages of the applications. Montage GALFA (5) from Cluster 2 is randomly chosen as the target application and the interference ratio between Montage GALFA (5) application and all other applications in the queue waiting to be scheduled to resources is calculated and presented in Table 4.

From the results, applications in Cluster 2 have the highest amount of interference with Montage GALFA (5 planes), with interference ratio ranging from 0.29 to 1.0. This is because they have more similar resource usage profiles than other applications, reflecting the contention for resources.

3. Experiments and Evaluations

① Performance Interference per Cluster

The proposed method is validated with the following illustrative experiment. The experiment also illustrates the performance interference in containers. For this experiment, it is assumed that only two containers, each running one application, can run on the physical resource available. In this experiment scenario, Montage GALFA(5 planes) application from Cluster 2 is paired and co-executed with CFD(512KB mesh) as C_0, Melt(3500 runs) as C_1 and another Montage GALFA(5 planes) application as C_2 from Clusters 0, 1 and 2 respectively under the same conditions according to the scenario in Figure 7.

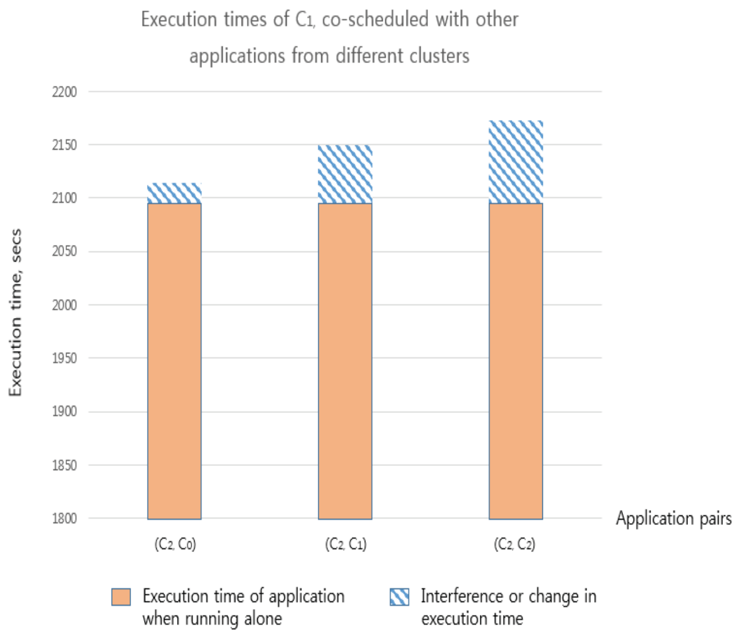

FIGURE 7: EXECUTION TIMES OF MONTAGE GALFA WITH CO-SCHEDULED APPLICATIONS

Figure 7 shows the changes in execution time of a new application, C_2 in Cluster 2, when co-executed with applications from different clusters, C_0, C_1 and C_2 respectively. From the above method, the interference ratio of the applications relative to C_2 is 0.00043, 0.01781 and 1, for C_0, C_1, and C2 respectively. The execution time of C_2 in each run is compared to the execution time of C_2 when executed alone in a container on the physical node, 2110 seconds. In this experiment, the differences in the execution time are considered as the amount of performance interference due to co-scheduling with another application.

From the results, C_2 experiences the most interference of about 63 seconds when executed with a similar application from the same cluster whilst C_2 experiences the least interference of 4 seconds when co-executed with C_0. This validates the hypothesis that there is performance interference among containers on the same physical resource as well as proves that the proposed interference ratio accurately predicts the interference among applications from different clusters. The results also show that, it is imperative to clearly understand the performance relationship between co-located applications during the co-scheduling of applications.

TABLE 5: RELATIONSHIP BETWEEN INTERFERENCE RATIO AND EXECUTION TIMES

Interference ratio, r	Normalized Execution time
0.0004	0.0019
0.2567	0.0190
0.5158	0.0013
0.7608	0.0000
1.0000	0.0299
Correlation	0.427

A further experiment to evaluate the relationship between the interference ratio and the execution times of applications is conducted. Selected applications with interference ratios of 0.00432, 0.2567, 0.51579, 0.7119481 and 1 respectively are co-executed.

The execution times of the applications are normalized according to equation 5 where ET_c is the execution time of an application when co-scheduled, and ET_a is the execution time of the same application when executed alone. From the results, the correlation between the interference ratio and the normalized execution times is calculated as 0.427. This shows a relatively positive relationship between the interference ratio and the ensuing change in execution times. Thus deploying the proposed method will facilitate proper co-scheduling of applications in order to mitigate interference and improve performance of applications.

② Interference for Different Scheduling Strategies

In order to validate the proposed interference-aware scheduling service, 5 applications of Montage GALFA(5 planes) and 5 Melt(3500 runs) applications are scheduled and executed using three scheduling strategies: bin-packing, random and the proposed interference aware scheduler which is based on spread strategy[30]. The proposed scheduling strategy is compared to the two other scheduling strategies.

The bin-packing strategy chooses resources based on the highest number of containers running on that resource whilst the random scheduler does not consider the number of running containers. The proposed method however, considers the interference between applications and schedules applications to resources with the least number of containers running on them based on the assumption that fewer containers correspond to less contention. Due to the differences in execution times of both applications, Melt(3500 runs) is run iteratively in 20 runs to cover the span of time for which Montage GALFA (5 planes) is executed.

Figure 8 describes the scheduling result of the interference-aware scheduling algorithm, as shown in Chapter III. For each scheduling method, the average execution times of Montage GALFA(5 planes) is calculated and the results are compared to the execution time of running the application alone, 2110 seconds.

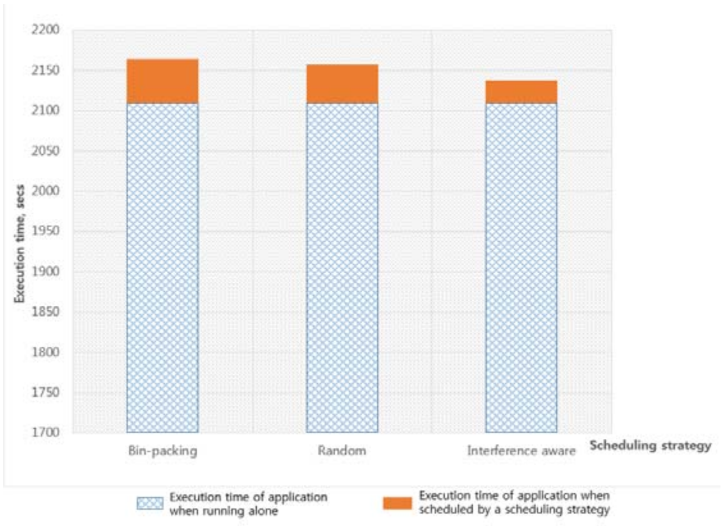

FIGURE 8: EXECUTION TIMES OF MONTAGE GALFA(5 PLANES) USING DIFFERENT SCHEDULING METHODS

From the results, the execution time of the proposed interference-aware method is the least at 2147 seconds whilst that of bin-packing strategy is the highest at 2164 seconds. This is because bin-packing strategy packs containers on the node with the highest number of containers without considering the contention for shared resources. Increased number of containers indicate a higher contention for shared resources. This also shows that the proposed method is able to mitigate interference.

A similar experiment is run with 1 Montage GALFA(5 planes) workflow, 1 Melt(3500) application as well as 1 Peptide(3000) application and co-executed them in three runs using three scheduling strategies: bin-packing, random and the proposed interference aware scheduler. Similarly, since the bin-packing strategy chooses resources based on the highest number of containers running on that resource, it schedules all the applications on the same node. The random scheduler and the proposed method however schedule both Montage GALFA(5 planes) and Peptide(3000) on the same node. The proposed method considers the interference ratio between the applications in making the scheduling decision and chooses Peptide (3000) which has a ratio of 0.00728 relative to 0.0181 of Melt(3500) applications. Due to the differences in execution times of both applications, Melt(3500) and Peptide(3000) applications are run iteratively in 20 runs to cover the span of time for which Montage GALFA(5 planes) is executed.

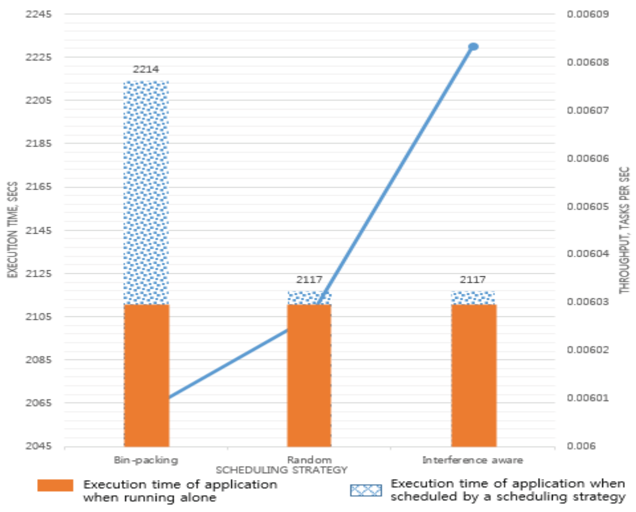

FIGURE 9: INTERFERENCE AMONG 3 APPLICATIONS USING DIFFERENT SCHEDULING STRATEGIES.

Figure 9 describes the scheduling result of the proposed interference-aware scheduling algorithm. For each scheduling method, the execution time of the workflow, Montage GALFA(5 planes), is obtained and the results are compared. From the results, the execution time of the proposed interference-aware method and the random method is the least at 2117 seconds whilst that of bin-packing strategy is the highest at 2214 seconds. This is because the bin-packing strategy schedules all the applications on the same node without considering the contention for shared resources.

Also from figure 9, the proposed method is able to improve the throughput of Montage GALFA application by 4.58% relative to the throughput when executed alone as well as mitigate interference.

③ Throughput Optimization

In the above scenario, the proposed method is able to improve the execution time for which the Montage GALFA(5 planes) application is executed. This translates to a throughput of 0.0060504 tasks per second relative to that of 0.006007 tasks per second when the application is run with bin-packing method for instance. However, Montage GALFA(5 planes) consists of 13 major tasks executed by different modules in the workflow during the execution process. This number of tasks is relatively few and thus does not adequately reflect the improvement in throughput obtainable by the proposed method. Thus, a simulation experiment using CloudSim[47], which considers applications with different tasks is conducted in this section to show the throughput improvement by the proposed service system. In this thesis, throughput T, is defined as the number of tasks completed during an execution, per the time of execution as seen in equation 5.

$$\text{Throughput, T} = \frac{\textbf{Number of tasks}}{\textbf{Execution time}} \qquad (5)$$

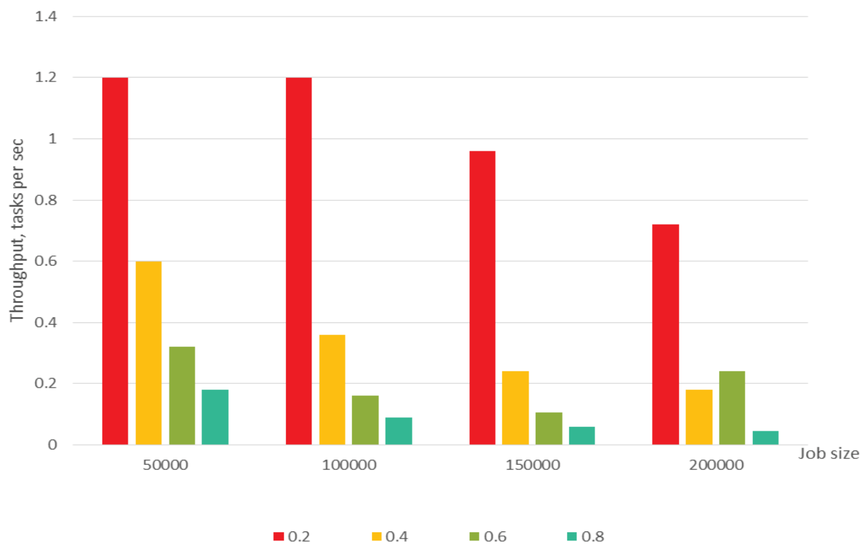

FIGURE 10: RELATIONSHIP BETWEEN INTERFERENCE RATIO, JOB SIZE AND THROUGHPUT

In the simulation scenario, different jobs with different number of tasks are executed and their throughputs are compared. This thesis sets the interference to 0.2, 0.4, 0.6 and 0.8 for these experiments. From the results, more tasks are executed when the interference ratio is at 0.2 as seen by the high throughput value. This adds to the fact that including interference-awareness in scheduling decisions helps improve the throughput and the performance of applications. The throughput however reduces as the job size increases due to the increased number of failed jobs as a result of the no-schedule policy of the proposed scheduling system.

V. CONCLUSION

In this thesis, an interference-aware scheduling service system that mitigates the problem of performance interference based on the results from clustering analysis of application characteristics is proposed. The proposed scheduler schedules containers to resources based on results from clustering analysis of the applications running on them in order to reduce interference from co-located applications. The proposed method is evaluated with corresponding experiments and it is compared with two other scheduling methods used in container clusters. From the results, the proposed method improves the performance of the target application as well as improves the throughput of the target applications. Also, the optimal interference ratio for maximized throughput as well as the optimal threshold value at which the execution time is minimal is evaluated with corresponding experiments. From the experiments, interference-awareness improves the execution times of applications and thus improves performance.

In the future, the application analysis would be expanded to include other metrics of the application such as I/O, Network bandwidth usage, data locality and number of instructions executed. With the results, the proposed method would more accurately predict the interference between applications.

VI.REFERENCES

[1] Adufu T., Choi J., and Kim Y., “Is container-based technology a winner for high performance scientific applications?” Network Operations and Management Symposium (APNOMS), 2015 17th Asia-Pacific

[2] Felter W., Ferreira A., Rajamony R., and Rubio J., “An Updated Performance Comparison of Virtual Machines and Linux Containers,” IBM Research Report, RC25482 (AUS1407-001), 2014.

[3] Li W., Kanso A., and Gherbi A., “Leveraging Linux Containers to Achieve High Availability for Cloud Services,” Proceedings of the IEEE International Conference on Cloud Engineering, 2015, pp. 76-83.

[4] Boettiger C., “An Introduction to Docker for Reproducible Research,” ACM SIGOPS Operating Systems Review - Special Issue on Repeatability and Sharing of Experimental Artifacts, Vol. 49, Issue 1, 2015, pp.71-79

[5] Memari N., Hashim S. J. B., and Samsudin K. B., “Towards virtual honeynet based on LXC virtualization” Region 10 Symposium, 2014 IEEE, pp. 496-501

[6] Bui T., “Analysis of Docker Security,” arXiv:1501.02967 [cs.CR],

2015.

[7] Varghese B., Akgun O., Miguel I., Thai L., and Barker A., “Cloud Benchmarking for Performance,” Proceedings of the 6th IEEE International Conference on Cloud Computing Technology and Science, 2014, pp.535-540

[8] Yoon H., Ravichandran M., and Schwan K., “Distributed Cloud Storage Services with FleCS Containers,” Open Cirrus Summit, 2011

[9] Nadeem F. and Fahringer T., “Predicting the execution time of grid workflow applications through local learning”, In Proceedings of the Conference on High Performance Computing Networking, Storage and Analysis, SC ’09, pages 33:1–33:12. ACM, 2009.

[10] Zhang Y., Sun W., and Inoguchi Y., “Predicting running time of grid tasks based on cpu load predictions”, In Proceedings of the 7th IEEE/ACM International Conference on Grid Computing, GRID ’06, pages 286–292. IEEE Computer Society, 2006,

[11] Ron C., Chiang H., and Huang H., “TRACON: Interference-Aware Scheduling for Data-Intensive Applications in Virtualized Environments”, IEEE Transactions On Parallel And Distributed Systems, 2011 International Conference.

[12] FitPredicates: Scheduling policy in Kubernetes, available online: https://github.com/kubernetes/kubernetes/blob/master/plugin/pkg/scheduler/algorithm/predicates/predicates.go (accessed on April 27, 2016).

[13] PriorityFunction: A scheduling policy in Kubernetes, available online:
https://github.com/kubernetes/kubernetes/blob/master/plugin/pkg/scheduler/algorithm/priorities/priorities.go (accessed on April 27, 2016).

[14] Marathon: A framework for Apache Mesos, available online:
https://mesosphere.github.io/marathon/docs/ (accessed on April 27, 2016).

[15] MacQueen J. B., "Some methods for classification and analysis of multivariate observations," in Proc. of the fifth Berkeley Symposium on Mathematical Statistics and Probability, L. M. L. Cam and J. Neyman, Eds., vol. 1. University of California Press, 1967, pp. 281–297.

[16] Kamp P. and Watson R. N. M., "Jails: Confining the omnipotent root", In Proceedings of the 2nd International SANE Conference, 2000.

[17] Potzl H., Linux-vserver technology, available online:
http://linux-vserver.org/Linux-VServer-Paper, 2004 (accessed on

April 27, 2016).

[18] Sun Microsystems. Solaris containers: Server virtualization and manageability, available online:
http://www.sun.com/software/whitepapers/solaris10/grid containers.pdf, (accessed on April 27, 2016).

[19] Alpern B., Auerbach J., Bala V., Frauenhofer T., Mummert T., and Pigott M., "Pds: A virtual execution environment for software deployment", In Proceedings of the 1st International Conference on Virtual Execution Environments, 2005.

[20] SWsoft. Virtuozzo server virtualization, available online:
http://www.swsoft.com/en/products/virtuozzo (accessed on April 27, 2016).

[21] Docker, available online: http://www.docker.com (access on April 27, 2016).

[22] Soltesz S., Potzl H., Fiuczynski M. E., Bavier A., and Peterson L., "Container-based Operating System Virtualization: A Scalable, High-performance Alternative to Hypervisors", EuroSys '07 Proceedings of the 2nd ACM SIGOPS/EuroSys European Conference on Computer Systems 2007, pp. 275-287

[23] Xavier M. G., Neves M. V., and DeRose C. A. F., “A Performance Comparison of Container-based Virtualization for MapReduce Clusters” Parallel, Distributed and Network-Based Processing (PDP), 2014 22nd Euromicro International Conference on, 2014, pp. 299-306

[24] Xavier M.G., Neves M.V., Rossi F.D., Ferreto T.C., Lange T., and DeRose C.A.F., “Performance evaluation of container-based virtualization for high performance computing environments.” In Parallel, Distributed and Network-Based Processing (PDP), 2013 21st Euromicro International Conference on, pages 233–240, Feb 2013.

[25] Memari N., Hashim S. J. B., and Samsudin K. B., "Container Based Virtual Honeynet for Increased Network Security" Information Technology: Towards New Smart World (NSITNSW), 2015 5th National Symposium on Date 17-19 Feb. 2015

[26] Vasić N., Novaković D., Miučin S., Kostić D., and Bianchini B., “DejaVu: Accelerating Resource Allocation in Virtualized Environments”, In ASPLOS, 2012.

[27] Kubernetes: An Open-Source Cluster Manager from Google, available online: http://kubernetes.io/ (accessed on April 27, 2016).

[28] Dharmesh K. Apache Mesos Essentials. Packt Publishing Ltd, 2015.

[29] Docker Swarm, clustering for Docker, available online: https://docs.Docker.com/swarm/ (accessed on April 27, 2016).

[30] Docker Swarm rescheduling policies, available online: https://github.com/Docker/swarm/blob/master/experimental/rescheduling.md (access on April 27, 2016).

[31] Aurenhammer F., and Klein R., "Voronoi Diagrams." Ch. 5 in Handbook of Computational Geometry (Ed. J.-R. Sack and J. Urrutia). Amsterdam, Netherlands: North-Holland, pp. 201-290, 2000.

[32] Tibshirani R., Walther G., and Hastie T., "Estimating the number of clusters in a data set via the gap statistic", J. R. Statist. Soc. B (2001) 63, Part 2, pp. 411-423

[33] Kodinariya T.M., and Makwana P.R., "Review on determining number of Cluster in K-Means Clustering", International Journal of Advance Research in Computer Science and Management Studies, Volume 1, Issue 6, November 2013.

[34] Cross validation method: available online: https://www.cs.cmu.edu/~schneide/tut5/node42.html (accessed on April 27, 2016).

[35] Elbow Method, available online:
https://en.wikipedia.org/wiki/Determining_the_number_of_clusters_in_a_data_set

[36] Di S., Kondo D., and Cappello, “Characterizing Cloud Applications on a Google Data Center”, 2013 42nd International Conference on Parallel Processing

[37] Alex N., “Interference Estimation in Cellular Systems” IJCA Special Issue on “Novel Aspects of Digital Imaging Applications” DIA, 2011

[38] Duan R., Prodan R., and Xiaorong L., “Multi-Objective Game Theoretic Scheduling of Bag-ofTasks Workflows on Hybrid Clouds”, IEEE transaction on cloud computing, vol .2 , no. 1, January-March 2014.

[39] Computational Fluid Dynamics, available online: http://www.cfd-online.com/Wiki/Main_Page (accessed on April 27, 2016).

[40] Melt, available online:
http://lammps.sandia.gov/doc/Section_example.html

[41] Peptide, available online:
http://scent.gist.ac.kr/downloads/tutorial/2012/1/Lammps_Tutorial_2

0120706.pdf

[42] Montage, available online: http://montage.ipac.caltech.edu/

[43] Peek et al., “The GALFA-HI Survey: Data Release 1”, The Astrophysical Journal Supplement, Volume 194, Issue 2, article id. 20, 13 pp. 2011.

[44] Valgrind, available online: http://valgrind.org/ (accessed on April 27, 2016).

[45] htop, available online: http://hisham.hm/htop/ (accessed on April 27, 2016).

[46] Jain I.S., Aalam M. A., and Doja M. N., “K-means clustering using WEKA interface”, Proceedings of the 4th National Conference; INDIACom-2010 Computing For Nation Development, February 25 – 26, 2010 Bharati Vidyapeeth’s Institute of Computer Applications and Management, New Delhi

[47] Rodrigo N., Calheiros, Ranjan R., Beloglazov A., DeRose C. A. F., and Buyya R., "Cloudsim: a toolkit for modeling and simulation of cloud computing environments and evaluation of resource provisioning algorithms," Software: Practice and Experience, 41(1), pp. 23-50, 2011.

ABSTRACT IN KOREAN

가상 환경에서의 응용 군집 분석을 이용한 간섭 고려 자원 스케줄링 기법

아두푸 테오도라 카푸이 아조
Dept. of Computer Science
The Graduate School
Sookmyung Women's University

컨테이너 기술과 같은 가상화 기술은 과학 응용의 실행을 위해 확장 가능한 온 디맨드 자원 프로비저닝을 제공하며 점점 더 많은 분야에서 사용되고 있다. 그러나 효율적인 성능 분리 계층 없이는 같은 자원을 사용하는 응용의 성능에 영향을 미치지 않는 것이 보장되지 않는다. 본 연구에서는 이를 해결하기 위해 응용 특성의 군집화 분석을 기반으로 간섭-인지 스케줄링 방법을 제안한다. 제안된 시스템은 공유 자원에 대한 경쟁을 줄이기 위해 응용 사이의 간섭의 양을 감지하여 동시 스케줄링을 진행한다. 실험은 다른 간섭인지 스케줄링 기법과의 비교를 통해 제안하는 방법이 대상 응용의 실행 시간을 줄일 수 있음을 보였다.

주제어: OS-레벨의 가상화, 컨테이너, 응용 프로그램의 특성, 간섭 인식, K-means, 간섭