Computers and Electrical Engineering 68 (2018) 119–139

Contents lists available at ScienceDirect

Computers and Electrical Engineering

journal homepage: www.elsevier.com/locate/compeleceng

Efficient data and CPU-intensive job scheduling algorithms for healthcare cloud☆

Prasan Kumar Sahoo[a,b,c], Chinmaya Kumar Dehury*,[a]

[a] Department of Computer Science and Information Engineering, Chang Gung University, Guishan 33302, Taiwan
[b] Department of Cardiology, Chang Gung Memorial Hospital, Linkou 33305, Taiwan
[c] Division of Colon and Rectal Surgery, Chang Gung Memorial Hospital, Linkou 33305, Taiwan

ARTICLE INFO

Keywords:
Cloud computing
Scheduling
Data-intensive
CPU-intensive

ABSTRACT

Cloud computing platform is used to improve the operational efficiency of business processes and to provide services to users. The fast growth of healthcare industry is shifting its traditional business model to cloud-enabled business model, which can fulfill the resource demand of different applications in healthcare industries. The job of healthcare system can vary from a simple patient record retrieval to a complex biomedical image analysis. On the other hand, the shared configurable computing, storage and other resources are provided to the users as a service over the Internet on rented basis. Although huge amount of resources are provided by the cloud to the healthcare systems to carry out complex time-consuming and data-intensive operations, scheduling of diverse healthcare applications onto large numbers of physical servers is an evolving issue, which needs to be addressed. In this paper, a scheduling framework is designed for the intelligent distribution of healthcare related jobs based on their types by taking advantage of existing heterogeneous distributed data center management solutions. This loosely coupled architecture works atop the existing solutions whose components can run in parallel on different nodes. The proposed framework for cloud computing can not only handle a variety of jobs but also can recover itself from any accidental crash.

1. Introduction

The flexibility and elasticity properties of cloud computing increase its demand and popularity among researchers, students, industries and others. Since, huge amount of data are produced every day, managing those data become cumbersome and is time consuming for many companies such as Microsoft, Yahoo, Google etc. The cloud computing environment provides different resources to users as a service by deploying different service models such as software as a service, platform as a service and infrastructure as a service [1].

The healthcare industry, one of the largest sector is taking advantages of the cloud computing environment in order to fulfill its exponentially growing resource demand. The healthcare system encompasses a wide variety of jobs ranging from manually identifying bone fracture to application of molecular medicine in personalized medicine. The current challenges force the modern healthcare industries to switch their business process to be fully automated and cost effective in order to compete with their rivals. Buying and deploying high-end computer system for different jobs is very expensive, which further required additional human resources. The cloud environment is the answer to the aforementioned problems, which can eliminate the need for huge capital

☆ Reviews processed and recommended for publication to the Editor-in-Chief by Associate Editor Dr. L. Bittencourt.

* Corresponding author.
E-mail addresses: pksahoo@mail.cgu.edu.tw (P.K. Sahoo), D0321009@stmail.cgu.edu.tw (C.K. Dehury).

https://doi.org/10.1016/j.compeleceng.2018.04.001
Received 17 May 2017; Received in revised form 31 March 2018; Accepted 2 April 2018
Available online 09 April 2018
0045-7906/

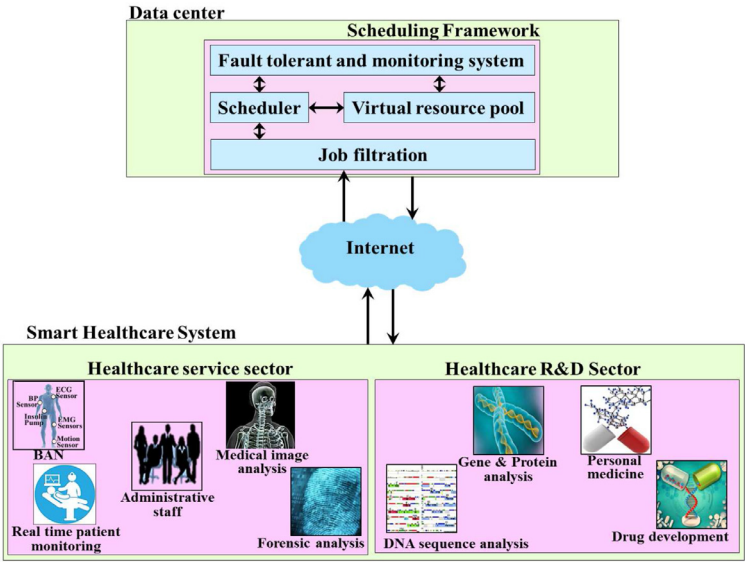


Fig. 1. Integration of cloud computing and healthcare system.

investment and additional human resources for computer maintenance.

The basic requirement of healthcare applications is huge number of high-performance computational nodes, which can be fulfilled by cloud. The underlined virtualization technology of cloud facilitates the users to create virtual version of the available physical resources such as servers, storage devices, networks or even the operating systems. Virtualization can be implemented in two ways, which are hypervisor-based and container-based virtualization. The VMs created by hypervisor-based virtualization technology consume more space than the containers. Docker [2] is one of the most popular platforms to setup container-based virtualization environment. The containers, also known as VMs, are created atop the kernel of the host OS. Before any job starts its execution onto any physical node, a set of resources such as CPU, memory, storage and network bandwidth are assigned in the form of Virtual Machine (VM) [3]. This VM abstracts the details of the physical node from any incoming jobs as it emulates the computing environment.

It is very crucial to manage huge amount of resources while assigning a large number of variety of healthcare related jobs to multiple computing nodes. The major issue is scheduling the jobs that are generated from hospital staffs, patients, researchers etc. Many researchers classify the scheduling problem into resource-aware and network-aware scheduling. Resource-aware scheduling mainly focuses on the scheduling of the jobs based on the resource requirement, such as storage, memory, CPU etc. by the jobs. On the other hand, network-aware scheduling mainly focuses on the scheduling of jobs based on the required network bandwidth. However, the scheduler must be efficient enough to handle different kinds of jobs irrespective of the type of jobs with the goal to maximize the utilization of deployed physical servers and the underline network infrastructure. In further sections, we will discuss the application of cloud in healthcare system, as shown in Fig. 1 including the research related to human body. The major contributions of our research can be summarized as follows.

- We apply computation and I/O data payload ratio to classify the incoming healthcare jobs.
- Two efficient scheduling algorithms are proposed exclusively dedicated for CPU-intensive and data-intensive healthcare jobs.
- In order to provide higher degree of fault-free environment for the VMs executing the healthcare jobs, a mechanism for monitoring and recovery of VM from failures is proposed.
- Implementation and simulation results show that our approach can significantly improve the resource utilization and jobs execution time over existing approaches.

The rest of the paper is organized as follows. The related works along with our motivation and goals are presented in Section 2. The integration of cloud computing platform and healthcare system is described in Section 3. Section 4 presents our proposed scheduling mechanism. Data type based job scheduling algorithms are designed in Section 5. The analytical model of our proposed framework is given Section 6. Implementation and simulation results of our proposed scheduling mechanism are described in Section 7 and concluding remarks are made in Section 8.

2. Related works

For the performance optimization of parallel and distributed systems, assignment of the resources to the scheduled jobs plays an important role. The problem of mapping of resources to the incoming jobs has been studied extensively in [4–6]. Several research articles have also been published considering the health care system in cloud environment and big data [7]. In [5], authors formulate

a model for task scheduling and the proposed PSO model focuses on optimization of total executing cost and the data transfer time. Though this paper claims to schedule data and CPU-intensive applications in an optimized way, applying all tasks to all processors and choosing the best fitness value among all particles could be a burden when it comes to a real-world cloud implementation.

The major issue with shifting the healthcare industry to cloud is confidentiality, integrity and availability properties of the patient data due to remote storage [8]. In [9], authors discuss the potential applications of cloud computing in research communities related to healthcare industry such as personalized medicine, genomic research etc. Considering the rural healthcare monitoring system authors in [10] proposes novel framework for medical systems which is explicitly designed in conjunction with cloud.

Authors in [11] propose a genetic matching algorithm to select a suitable cloud service provider taking the user requirements and properties of cloud provider into account. Authors consider the execution of DNA sequencing workflows in multiple clouds environment. As the healthcare industry generates massive amount of data, it is very crucial to revisit the existing solutions that are responsible for processing and analyzing the healthcare big data in cloud computing environment, as discussed in [12]. Hadoop cluster can be considered as a candidate solution for processing, storing and analyzing the healthcare big data in cloud environment. However, the major issue with the Hadoop is its inability to handle the streaming data that are generated from patients.

An idea of prioritizing the virtual machine is presented in [6], where all VMs are divided into two tiers based on their priority and are executed on a node either in foreground or in background. A task-level scheduling algorithm is proposed in [13] with budget and deadline constraints. Authors focus on two basic optimization problems with respect to the fixed budget and deadline of the job execution. To optimize the total cost, problem of dynamic resource provisioning problem is formulated in [14] with the objective to minimize the total number of active VMs. However, applications of data-intensive jobs to the proposed scheme is a major drawback.

Considering server inlet temperature constraint and taking the benefit of electricity price variation across time and location, a thermal-aware scheduling algorithm is proposed in [15,16]. Authors in [17,18] propose a scheduling mechanism for real-time aperiodic independent tasks. However in [17], implementing the scheduling mechanism under the assumption that the accurate task execution time is known is nearly impossible. Google's MapReduce [4], Yahoo's Map-Reduce-Merge and Microsoft's Dryad are the examples of customized data processing framework focusing on massive data read/write operations. Another parallel data processing framework focuses on I/O or data-intensive pipelined workflow known as Nephele [19]. All tasks may be completed in different time instance in a single level. This may cause an instance to go idle when other tasks in the same level are still running. As presented in [20], the jobs with skewed data are handled by introducing an approach that works in the reduced phase of MapReduce framework. The proposed approach suffers from the limitation of handling compute-intensive jobs in MapReduce framework. By considering Map-Reduce environment, authors in [21] propose techniques for communication optimization in map-reduce applications by scheduling the virtual machines near to the reduce task, which may not guarantee early completion of the task.

For multi-job MapReduce workloads, a dynamic scheduling technique is proposed in [22] with the model to predict the remaining completion time of the job. Since the prediction is based on historical information, this may not be able to predict accurately during the inception of the job execution. A novel scheduling algorithm is proposed in [23] for the deadline constrained service workload and dynamic auto-scaling of resources on IaaS cloud environment. Though, this enables the cloud service provider to utilize the resources efficiently, however it is not clear how to handle the jobs when the privately provisioned resources are in full utilization. An extensive comparison of the aforementioned related works is presented in Table 1.

2.1. Motivation

In the recent past, various legacy software tools have been developed to smoothen the healthcare business process. These software tools facilitate the patients and doctors to summarize the past health record, analyze various health-related parameters to predict the potential diseases. These software tools are normally used within the premises of a healthcare organization, which requires huge amount of hardware and software resources. Various third-party vendors take advantage of the cloud computing environment and offer the healthcare related legacy software tools on a rented basis. In order to provide software as a service, service providers employ various algorithms such as scheduling and resource management in the cloud environment. These algorithms guarantee the demand of cloud-based healthcare software tools without losing the revenue of the cloud service provider.

Now-a-days patients are fitted with multiple sensors to collect various bio-medical signals. Besides, healthcare industries are equipped with various devices to collect patients data in form of digital images captured by x-ray, ultrasound, MRI, optical imaging technology etc. Analysis of bio-medical images in cloud environment requires huge number of high-performance computing nodes. The images can be processed either in single or multiple computing nodes in parallel manner. Slicing the bio-medical images and thereafter analysis of the sliced images concurrently in multiple nodes can reduce the time effectively. Besides, number of computing nodes can be scaled up and scaled down dynamically. Similarly, the bio-medical signals such as ECG and EMG can be processed in cloud environment.

Although varieties of scheduling algorithms and data processing frameworks have been proposed, it is observed that cloud computing environment for the healthcare industries needs special attention to meet the requirement of emergency data processing. Scheduling of different types of jobs such as DNA sequencing and analyzing the real-time sensors data generated from the patient is one of the major issues that needs to be addressed. Precisely, it is found that the actual resource requirements are not taken into consideration to meet the deadline of the job execution in order to maximize the utilization of physical resources. For example, if a task of a job needs 100 units of computation capacity and 200 units of network bandwidth, the respective VMs should also be configured with exact amount of resources in order to meet the deadline. Besides, based on the configuration, the VMs can either be homogeneous or heterogeneous. Homogeneous VMs may lead to either inefficient utilization of resources or unavailability of resources, which results in revenue degradation and poor QoS.

Table 1
Comparison of related works.

References	Job types	Jobs classified or not	Resource reconfiguration	Resource utilization	Name of scheduling algorithm	Findings
[11]	Healthcare workflows	No	No	No	HU-GA matching algorithm	Reduces execution costs meeting the SLAs.
[14]	Compute-intensive applications	Yes	Yes	Yes	DIPLDP Algorithm	Minimize the total computing time.
[15]	All types of jobs	No	No	No	GreFar Algorithm	Optimizes energy cost.
[17]	Real-time, Aperiodic and independent jobs	No	No	Yes	PRS algorithm	Improve Resource utilization, reduce energy consumption.
[19]	All types of Jobs	No	Yes	Yes	Nephele Framework	Reduces processing time and cost.
[20]	MapReduce Job	No	No	Yes	FP-Hadoop	Reduces total execution time of MapReduce jobs.
[21]	MapReduce Job	No	No	Yes	LA-VMM algorithm	Improving performance of MapReduce programs.
[23]	All types of Jobs	No	Yes	Yes	Deadline constrained scheduling algorithm	maximize the number of workloads meeting their deadline.

Some healthcare applications may have higher demand of computing resources such as analyzing the bone fracture from digital x-ray images, which are deemed to be CPU or compute-intensive applications. On the contrary, applications those require massive amount of data and dedicate most of their processing time to the data for I/O operations are known as data-intensive applications. In case of data-intensive applications the CPU utilization is less. Taking the characteristics healthcare jobs into account, the key jobs scheduling problem can be of two types. First, the healthcare jobs may vary from simple data retrieval to complex biomedical image analysis. Based on the variety, it is necessary to redesign the scheduling algorithms for the healthcare services in order to minimize the job execution time. Secondly, utilization of resources while scheduling variety of jobs is inefficient as most of the scheduling algorithms do not consider the size and nature of the jobs itself.

Both of the above-mentioned applications have their own goals. For data-intensive applications, the goal is to minimize the movement of data from one node to another, whereas for CPU-intensive applications, the goal is to schedule the job to high-performance computing cluster in order to minimize the total execution cost. In order to achieve these two goals, both classes of applications need to be treated separately to simplify the scheduling process, which motivates us to design the scheduling framework and algorithms for both data and CPU-intensive tasks. Based on our motivation, the main goals of our work can be summarized as follows.

- To classify the incoming jobs generated by patients, doctors and researchers based on the computation and I/O data payload ratio.
- To minimize the total execution time for compute-intensive jobs such as cloud-based medical image and DNA analysis.
- To minimize the delay caused due to insufficient network resources in case of data-intensive job such as DNA sequencing.
- Utilize the resources while the virtual machines are busy for I/O operations.
- Reconfigure the virtual machines to meet the job execution deadline based on the jobs types.

3. Integration of cloud computing with healthcare system

The jobs in healthcare industries can be divided into two types. Firstly, the jobs related to provide services to the patients and secondly, the jobs related to research and development sectors. As the hospital industries are moving their business process to cloud-enabled solutions, diverse users access the services from the remotely deployed software solution platforms over Internet. This enables the hospital users and staffs to access the patient related information and other hospital related information at any time and anywhere. On the other end, the cloud computing environment in a data center is empowered with huge number of physical servers, provide physical resources in the form of virtual resources based on the demand of hospital. The applications of cloud computing in healthcare system is shown in Fig. 1.

3.1. Smart healthcare system

The smart healthcare system comprises diverse applications or jobs such as body area network of patients, real-time patient monitoring systems, biomedical image analysis, forensic analysis, gene and protein analysis, drug development, DNA sequence analysis and administrative jobs as shown in Fig. 1. The applications within a smart healthcare system can be classified as healthcare service sector and healthcare research and development (R&D) sector. In healthcare service sector, the hospital staffs are mainly involved in providing services to the patients, whereas the researchers are mainly involved in wide research areas such as drug development in healthcare R&D sector. Doctors and hospital staffs are empowered with cloud-enabled software tools, which enable doctors to monitor corresponding patients remotely. The patients are fitted with multiple sensors to monitor different health-related parameters such as glucose level, blood pressure, Hemoglobin level and ECG sensors. The raw data generated by sensors in body area network are transmitted to cloud over Internet where the raw data are preprocessed to the readable format and are stored for future analysis. The uploaded data can further be monitored remotely by the doctors and the automated sophisticated software tools can be applied to monitor the patient health on real-time basis. This significantly leverages the concept of virtual care and telemedicine. Cloud-based software tools are developed to support and simplify the doctors' decision-making process. For example, researchers are on progress to process and analyze the digital medical images in order to identify the diseases. Digital X-ray images can be analyzed with sophisticated resource-intensive cloud-based software tools to identify the bone fracture or displacement or to identify the tumors. Further, for the administrative staffs, the traditional healthcare applications can be modified and deployed in cloud environment to access and generate the patients' health report.

The modern healthcare industries focus on research and development of existing drugs and healthcare systems. The research area related to healthcare system is vast such as drug development, personalized medicine, gene and protein analysis, DNA sequence analysis etc. Patients genes are sequenced in order to evaluate the risk of genetic disease. In DNA sequencing, a large number of databases are involved in the study of evolution of different organisms. The resource demand of aforementioned healthcare applications plays an important role as the entire applications are fully dependent on the amount of resources provided. Considering the example of real-time patient monitoring, the computing resources that are provided must be enough to process and analyze the raw data with no delay. Lack of computing resources for monitoring the patients' data leads to failure of on-time processing. Further, in the healthcare R&D sector, large databases are stored in a distributed manner. In order to carry-out massive search and compare operations in such large databases, network bandwidth is one of the major resource requirements. Based on the resource demand, the jobs can be categories into two types: data-intensive jobs and CPU-intensive jobs. In data-intensive jobs, mainly large databases are involved that are distributed among multiple storage servers such as protein mass spectrometry, which refers to the study of proteins. In DNA sequence analysis, thousands of gene expressions are compared. Such application needs huge database of gene expressions

that is distributed over multiple storage servers. On the other hand, the CPU-intensive jobs demand high-end computing nodes. In healthcare system, the body area network of a patient body comprises multiple sensors to gather the healthcare parameters of human body. Those sensor-generated data need to be uploaded to the cloud for further analysis. Complex and time-consuming image processing algorithms are designed to analyze specific disease in human body. Such algorithms are highly parallel in nature and therefore can be assigned to multiple computing nodes for parallel processing. Here, computing node refers to as one virtual machine, which is responsible for carrying out the execution of job.

3.2. Cloud computing in healthcare system

The cloud environment is introduced with the responsibility to provide virtually unlimited computing, storage and network resources to the healthcare system. Its flexibility to scale up and scale down the resource encourages the healthcare system to adopt the cloud environment. Though, cloud promises to facilitate virtually unlimited resources to healthcare system, however improper handling of jobs that are arriving to cloud and inefficient management of resources in cloud may fail to comply with the requirements of healthcare industries. Hence, healthcare cloud must provide a dedicated framework in order to handle the aforementioned two types jobs.

In this paper, we are emphasizing on scheduling of the applications or jobs based on its classification. As a result, we propose a novel scheduling framework, which not only provides a mechanism to schedule the incoming jobs efficiently, but also provides a fault tolerant mechanism to execute the jobs without any interruption caused due to failure of the virtual computing nodes. Henceforth, the term *application* is referred to as a *job*.

It is to be noted that healthcare jobs are different from other applications and therefore need special attention as compared to other kinds of jobs due to its variety, size, resource requirement, priority and job completion time. As shown in Fig. 1, healthcare jobs may vary from retrieving single record of a patient to the very complex DNA sequence analysis. Some jobs need to be scheduled without any delay with higher priority, referred to as emergency jobs. Though such emergency jobs may be small in size with less execution time, however they need to be scheduled first. Further, CSP may receive diverse jobs in the sense that some jobs need negligible amount of resource whereas some others may need multiple VMs to execute. Based on the size of the job, the execution time may also vary from few milliseconds to more than one hour. Such diverse jobs need to be processed by the CSP using single framework essentially by analyzing each incoming job.

Though most cloud jobs can be classified into CPU and data-intensive, however, in healthcare applications, the jobs need to be classified taking negligible amount of time. With the objective to classify the jobs in negligible amount of time, we have considered only the CPU and data payload of each job as discussed in Section 4.1. Two scheduling algorithms are proposed here for data and CPU-intensive jobs as described in Section 5. Further, the virtual machines are created and reconfigured from the virtual resource pool in order to meet the actual requirement of the jobs and manage the physical resources efficiently. The fault tolerant and monitoring system provide an environment to execute the job with no interruption. The complete framework of our proposed scheduling mechanism is presented in Section 4.

4. Proposed scheduling framework

The basic idea of the proposed scheduling mechanism is to divide the incoming jobs into two pools: (a) data-intensive jobs and (b) CPU-intensive jobs. Both the pools of jobs are handled and scheduled separately to achieve the aforementioned goals. data-intensive applications are mostly dedicated to huge amount of data processing, where the computing nodes are mostly busy in data read or write operation. data-intensive applications are used in big data environments ranging from national security to individuals' location tracking and from analysis of astronomical images to medical image analysis. From the survey of related works as given in Section 2.1, it is found that few of the mechanisms are focusing only on scheduling data-intensive jobs, which may not perform well in scheduling CPU-intensive jobs. Based on the targeted environment, different factors are needed to be taken into account while scheduling the data-intensive tasks such as network related resource requirement, prioritization of the tasks based on the deadline or urgency level, prioritization of virtual machines based on the amount of allocated resources etc. In below we will provide some examples on CPU-intensive and data-intensive healthcare jobs.

Today's data-driven healthcare scientific research processes vast amount of varieties of real-time as well as non-real-time scientific data [17]. In order to process the non-real-time healthcare data that are stored in distributed servers, large number of virtual computing nodes are required. Considering the fact that increase in number of computing nodes increases the energy consumption and cost, the scheduling algorithm must schedule the jobs and the virtual nodes so that the underline resources are utilized efficiently. Hence, prioritizing virtual nodes and the jobs are essential.

Fig. 2 shows the abstract architecture and interaction of various system components of the proposed framework. The proposed framework consists of five major components: (I) Job filter, (II) Scheduler or job dispatcher, (III) VM Auditor, (IV) VM repository, and (V) Fault Monitoring and Recovery System (FMRS). This framework focuses on both data-intensive and CPU-intensive jobs. data-intensive jobs are those who require huge number of data for read/write operations rather than computation such as mining entire user's database while conducting a survey. On the other hand, CPU-intensive jobs are defined as the jobs those require a large amount of computational power rather than the data read/write operations such as extracting raw data from the scanned reports, image processing etc. In order to make the framework highly customizable to fit into any data center and policies with the heterogeneous physical servers, most of the components are made plug-able.

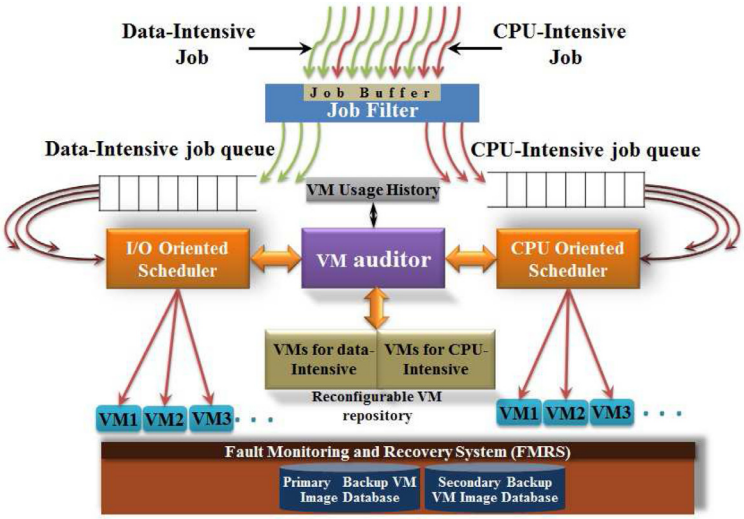


Fig. 2. The abstract model of the proposed scheduling framework.

4.1. Job classification

In order to assign the healthcare jobs to suitable virtual machines, the jobs are first analyzed and categorized into CPU or data-intensive job by "Job filter" component. In general, the arrival rate of the jobs may vary from small number of jobs to thousands of jobs per unit time. It is assumed that the jobs are independent and hence can be considered as classifiable. To handle such situation "Job buffer" component is introduced to support job filter component.

The responsibility job filter component is to filter each healthcare job by analyzing the type of request efficiently without taking significant time. Classification of a job can be done by analyzing its CPU payload and data payload. Though various classification methods exist, classifying the jobs by analyzing the CPU and Data payload makes the classification process easier and faster. Furthermore, this will also significantly reduce the classification time of the incoming jobs. The efficiency of such simple and faster classification method can be realized when huge number of healthcare jobs arrive at higher frequency. Here, CPU payload refers to time dedicated to different mathematical operations like numerical value comparison, division etc. Data payload refers to the data read or write from/to the hard disk or sending and receiving of data packets over the network. Compute-Data Tradeoff Threshold (CDTT) is used to decide if a job is data or CPU-intensive by cloud service provider. The CDTT value always lies in between 0 and 1, which represents the recommended percentage of total execution time of a job in order to consider the job as a CPU or data-intensive job. For example, CDTT value 0.5 indicates that 50% of the total execution time of the job is dedicated to data read/write operation. This value can be modified by the service provider as per requirement. A job is said to be CPU-intensive if the following condition holds.

$$\frac{Data\ Payload(t)}{Total\ Execution\ Time(t)} < CDTT. \tag{1}$$

The ratio of data payload and total execution time is calculated with respect to time *t*. If the resultant value is less at time *t*, the job is more likely to be involved in data reading or writing from/to hard disk or over the network. In other words, a job can be classified into either data-intensive or CPU-intensive by examining its compute to data ratio as in [24]. In [24], authors refer to data and CPU-intensive jobs as stabilization and accelerate mode, respectively. Authors considered the data transfer time from one cloud to another. However, the data transfer time from the I/O port to the secondary storage device and data transfer time among physical machines over the network is included in our classification method. After filtration, jobs are redirected to their respective queue. Based on the requirement, two queues are implemented for both CPU and data-intensive jobs. This filtration process is made easy to achieve our goal where high-performance computing servers should be assigned to execute the CPU-intensive jobs and servers with high network bandwidth must be engaged to execute data-intensive jobs.

Job Buffer: it is assumed that the job filter module is designed in such a way that it can filter massive number of jobs and can redirect them to their respective queues. Job buffer component stores the jobs when arrival rate is higher than the filtration rate. Filtration rate refers to the number of jobs filtered per unit time.

4.2. Job dispatcher

Job Dispatcher, Virtual Machine Repository, and Virtual Machine Auditor are all together responsible for receiving the classified healthcare related jobs. Scheduling of these jobs onto suitable virtual machines equipped with required healthcare software tools is highly essential. In the following subsections, we describe the individual responsibilities.

Job Dispatcher: Based on the algorithm presented in Section 5, the order of job execution is decided. For each queue, one dedicated scheduler needs to be designed in such a way that the utilization of the resources can be optimized without compromising the execution deadline of jobs. When a large numbers of small jobs arrive with the requirement of large number of small instances such as large number of patients are accessing previous month health report, they are redirected to the CPU-intensive queue. Two scheduling algorithms dedicated to CPU and data-intensive jobs are designed and are presented in Section 5. Those two scheduling algorithms send request to the VM auditor for respective types of VMs. For example, for CPU-intensive jobs, it is more important for the VMs to have enough computation capabilities rather than the network bandwidth. On the other hand, for data-intensive jobs, network bandwidth resource plays an important role rather than computation power in configuring a VM. For both schedulers, it is assumed that the VM reconfiguration incurs trivial cost in terms of time.

Virtual Machine Repository: All VMs are kept in a repository called reconfigurable VM repository. These VMs are the bridge between the physical server and user. Each VM in VM repository can be reconfigured to meet the requirement of job. For example, configuration of a VM for CPU-intensive jobs is not similar with VM configuration for data-intensive jobs. The dissimilarity is in terms of memory, processing capabilities, network bandwidth and storage. The repository is divided into two pools. One is for execution of data-intensive jobs and another is for CPU-intensive jobs. All VMs are controlled by the VM auditor component.

Virtual Machine Auditor: It plays an important role in configuring and allocating a virtual machine for which VM auditor must be aware of the cloud environment such as available VMs, their types as well as the cost. When a scheduler schedule some jobs, it needs to make sure that the required number of VMs are in allocation stage and are not busy. Insufficient numbers of VMs in allocation stage make the scheduler to send request to VM auditor for allocation of required numbers of VMs. VMs are assigned to the respective scheduler to start executing the incoming jobs. If the required number of VMs cannot be allocated to a specific type of job due to unavailability, different types of VMs should be leased to reconfigure and allocate them. For instance, if there is unavailability of a VM for the CPU-intensive job, the VM for the data-intensive jobs should be leased and reconfigured, which can be released after its service is completed. In addition to the above-mentioned functionalities, VM auditor is equipped with another sub-component known as *VM usage history*.

VM usage history plays a major role in managing VM repository. This helps VM auditor to take decisions such as when and how many VMs should be reconfigured for the upcoming jobs. Besides, VM usage history keeps track of the information about each VM such as VM ID (unique ID of each VM), type, location (e.g. VM belongs to which processing node), configuration/attribute of VM (e.g. speed, number of Hard Disks), current status, usage history of each VM, current state such as amount of memory used/free, disk capacity used/free, processor cores utilization, network bandwidth usage etc. From the historical data, many factors can be predicted such as when, how many VMs should be assigned to which type of job, when data-intensive workload can be more than the CPU-intensive workload and vice-versa. From this prediction, VMs can be reconfigured prior to increase in specific type of workload in the data center. This may increase the system performance by saving the reconfiguration time. Not only this, many future abnormal activities in the data center can be ignored before it actually happens. For example, a VM can be restored before it fails to execute the assigned job. This can be achieved by frequent monitoring of its success rate. In this context, success rate refers to as the successful completion of job execution without any complication from the VM itself.

For the sake of clarity, we have considered the scenario where a single data center equipped with High-Performance Computing (HPC) cluster handle a set of variety of applications forming a work set. We have taken the following applications into account:

DNA sequencing application: Determining the precise order of four chemical building blocks within a DNA molecule is referred to as DNA sequencing. Software tools designed for this application require huge I/O operations that makes it a data-intensive application.

Potential disease prediction: This involves analysis of a large number databases containing medical history of numerous patients and prediction of potential diseases.

Automatic biomedical image analysis: Automatic biomedical image analysis involves detection of any abnormality in human body using powerful image analysis algorithms. As the biomedical images are larger in size, it requires multiple high-performance virtual computing nodes to slice and analyze the data in parallel.

Patient health record statistics: Patients may send request from their hand-held devices to check the progress of their health from the inception of medication.

Personalized medical application: The health-related parameters of a patient needs to be analyzed efficiently for suitable medication and therapies that needs each individual's genetic content and other molecular analysis.

In general, different schedulers like FAIR scheduler, Torque, Condor [25] and other priority scheduler [21], schedule the jobs based on different factors such as priority, order of arrival, data locality etc. All schedulers treat different classes of jobs as identical. In this case, all five applications are identical for the above-mentioned scheduler. Hence, some jobs need to be assigned to the HPC clusters during the scheduling process. At the same time, some jobs need to be assigned to the VM instances having less computing capacity. Normally, user has to give this information about the required VM type.

In the proposed framework, let the job be scheduled by respective scheduler based on its type. On arrival of above applications, the proposed framework divides them into two classes. DNA sequencing jobs and potential disease prediction jobs are sent to the queue of data-intensive scheduler, whereas automatic biomedical image analysis, patient health record statistics and personalized medicine jobs are sent to the CPU-intensive scheduler. In this context, this classification of processes does not demand any extra information from the user. Since, DNA sequencing application and potential disease prediction jobs require a lot of memory operations like accessing huge number of files, the VM does not require to be configured with high computing capacity. Hence, they are assigned to specially configured VM instances after their execution, which is decided by the algorithm designed for scheduling the data-intensive jobs. Similarly, other applications are scheduled by another algorithm, which is designed for CPU-intensive jobs. Assigning suitable VMs for different class of application is the responsibility of the VM auditor. Both scheduling algorithm can run

concurrently on different nodes.

The proposed framework has three major advantages.

- Loosely coupled: Coupling refers to as the degree of direct knowledge. In other words, one component has little knowledge of another. For example, scheduler only schedules the jobs present in the queues and it has no knowledge about the filtration process.
- Highly parallel: The parallelism can achieve not only in job execution by running them concurrently on different nodes, but also the components of framework can run in parallel on different nodes. For instance, Fault Monitoring and Recovery System (FMRS) job filter can run independently on different nodes. This is possible because of lose coupling.
- Plug-able component: The property of lose coupling possesses another advantage, i.e. plug-able. This feature enables the users to add or replace any component at any time with newer improved technique. For example, another scheduling algorithm can be added in future for a new class of jobs, which can run on different nodes.

4.3. Fault monitoring and recovery system

Before discussing the working procedure of this component, let the necessity of this component be discussed. The requirements of CPU-intensive and data-intensive jobs in different scenarios such as healthcare service sector and R&D sector are different. However, the similarity in both the classes is that the assigned virtual computing node may take longer time to finish the execution of jobs. On the other hand, some computing nodes may be engaged in analyzing and exchanging the huge amount of stored data, which may take hours to finish the job. In such scenario, all virtual computing nodes must analyze the data without any failure. In order to facilitate a fault-free cloud environment, we have incorporated another component, called Fault Monitoring and Recovery System, which is discussed in the next sections.

Fault Monitoring and Recovery System (FMRS) monitors if any fault occurs in any virtual machine and recovers the virtual machine from the last recovery point. The goal of this component is to bring the job execution back to the running stage from any kind of failures. Failure of the VMs may occur in either parallel or in sequential order. Since, multiple VMs run concurrently, one or more VMs may encounter failure at any particular time. Further, a single VM may fail multiple times in a sequential order. Those parallel and sequential failures must be handled by FMRS component. It is assumed that service provider takes care of the failure on hardware level or other hosting applications. Each replica sent by the virtual machine to the FMRS is known as a Recovery Point of the respective virtual machine and is associated with the respective VM ID with current time stamp. It consists of two modules to keep the replicas of all virtual machines in different time instances. Both modules are referred to as the backup databases of the virtual machines.

Primary Backup VM Database: This Database is responsible for storing the recent replicas of all active/running VMs. We assume that an abnormally terminated/crashed VM can resume its normal execution from its most recently stored recovery point in primary backup VM database.

Secondary Backup VM Database: In order to avoid any failure on the primary backup VM database, FMRS also maintains another backup database called Secondary Backup VM Database.

Fig. 3 illustrates the interaction between FMRS and individual virtual machine. Each time when a virtual machine starts its execution and becomes active from the idle state, it should send its own replica with current time stamp and its own VM ID in a specific time interval. Each time when a VM sends its own replica, FMRS takes two actions. Firstly, the replica having same VM ID is moved to the secondary backup VM database from primary backup database and the previous replica/recovery point from primary database is deleted. Secondly, the new replica is inserted into the primary backup database.

If any failure occurs in the primary backup database, FMRS restores it by taking the images from the secondary backup database. At the same time, if FMRS detects any fault in a VM, it will try to resume the execution process from the recent recovery point having same VM ID from the primary database. Furthermore, if VM cannot resume its execution process from the replica of the primary database, the backup instance from secondary backup database will be considered to resume the job execution. In this context, we note that no application source code is required for communication between the VMs and fault monitoring and recovery system. The secondary backup VM database is maintained for two reasons: (1) this will help us to resume the VM from any abnormal functionality, if it fails to resume from its primary backup database. (2) Based on the first reason, we can employ mechanism to store all VM images from the inception of the execution of a task. But this may require high volume of storage space. In order to utilize the available database storage efficiently, all related backup data is removed from both primary and secondary backup database by FMRS after getting the termination signal from the scheduler. Between adjacent backup operations, the time duration must be balanced. Very frequent backup operations may keep the whole system busy in taking backup of the virtual machines and the actual job execution may be interrupted. On the contrary, large time gap between two backup operations may not provide restore point to resume the job execution. The component must be implemented in such a way that the normal execution of the jobs should not be interrupted by any backup operation.

5. CPU and data-intensive scheduling mechanism

As discussed earlier, the jobs are classified into two groups. They are CPU-intensive jobs and data-intensive jobs. It is assumed that the real-time performance can be achieved without violating the deadline of the jobs by providing the required amount of resources to the corresponding VMs irrespective of the type of the jobs. To schedule both healthcare related CPU and data-intensive jobs, two

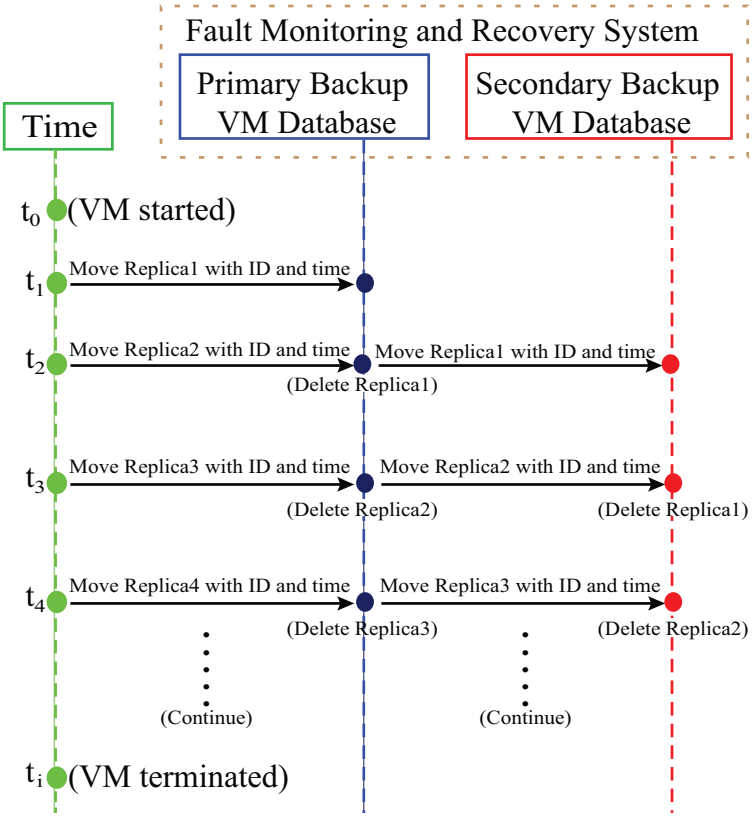


Fig. 3. Time-Sequence diagram for interaction between individual VM and FMRS.

dedicated scheduling algorithms are presented here. Let us consider a data center that facilitates the healthcare companies for hosting clouds environment with heterogeneous servers. Let N be the total number of jobs coming to the cloud at a particular instance of time. N^c represents the set of CPU-intensive jobs, whereas N^d represents the set of data-intensive jobs. It is assumed that N^c and N^d are disjoint sets. Hence, N can be formulated as $N = |N^c| + |N^d|$. It is assumed that CSP may receive emergency jobs, which need to be scheduled without any delay. These emergency jobs are considered as high priority jobs and are given higher preference during scheduling. It is assumed that clinicians and the end users decide if a job is emergency one or not. Let α be the boolean variable that indicates if a job is emergency job. The value of $\alpha = 1$, if the job is emergency one, otherwise it is set as 0. All the emergency jobs are classified as CPU-intensive jobs and are scheduled using CPU-intensive scheduling algorithm. This is due to the fact that all data-intensive jobs are scheduled in conjunction with a secondary task that runs in the background within the same VM. Scheduling the emergency job in conjunction with another secondary task may experience delay that may occur due to task switching within the VM. Let, V be the set of Virtual Machines (VMs) that are dedicated to execute different healthcare related jobs. VMs are of two types based on their configuration. Firstly, VM_c, the set of VMs that are configured to execute CPU-intensive jobs. Secondly, VM_d, the set of VMs that are configured to execute data-intensive jobs. The goal of the proposed mechanism is to minimize the average execution cost in terms of time and data transfer time, which are represented as C_c^{ec} and C_d^{ec}, respectively.

In the proposed scenario, it is assumed that the data center is equipped with k number of heterogeneous servers represented as set $S = \{s_1, s_2, s_3, ...,s_k\}$. Let $\delta_{core}(s_i)$ and $\delta_{mem}(s_i)$ be the amount of currently available cores and memory at server s_i, $1 \le i \le k$. As discussed earlier, N^c and N^d represents the set of CPU and data-intensive jobs, which need to be scheduled, respectively. For each job j, $j \in \{N^c \cup N^d\}$, the set of dependent tasks and their relationships are represented as Directed Acyclic Graph (DAG) G, where $G = (\widehat{V}, \widehat{E})$. Here, $\{\widehat{V}\}$ represents the set of tasks and $\{\widehat{E}\}$ represents the set of dependencies among the tasks. A directed edge e, $e \in \{\widehat{E}\}$ from node A to B indicates the dependency of node B on A. Considering above nomenclature, algorithms for CPU-intensive job scheduler and data-intensive job scheduler are presented below.

5.1. CPU-intensive job scheduler

Before scheduling any task, scheduler picks up a healthcare related job consisting of multiple tasks. It is very essential to get information regarding the physical resources such as number of cores and amount of memory available to distribute all tasks uniformly. The set of the jobs is first sorted in descending order based on the priority value as mentioned in Line 1 of Algorithm 1. The job with higher priority value or the emergency jobs are given higher preference during the scheduling. Let, Δ_{core} and Δ_{mem} be the total amount of cores and memory currently available in all physical servers as calculated in Line 4-5.

Data: N^c: the set of CPU-intensive jobs
$S = \{s_1, s_2, ..., s_k\}$ the set of physical servers
Sort N^c based on the priority value (α) of each job j in descending order;
foreach *Job* $j : N^c$ **do**
    /* Extract the high priority or emergency job first. */
    Prepare DAG $G = (\hat{V}, \hat{E})$ for current job j;
    Calculate $\Delta_{core} = \sum_{i=1}^{k} \delta_{core}(s_i)$ /* Available number of Cores */
    Calculate $\Delta_{mem} = \sum_{i=1}^{k} \delta_{mem}(s_i)$ /* Available number of memory */
    L = The total number of levels in DAG ;
    foreach *level* l:L **do**
        $TS(l)$ =set of tasks in level l ;
        $\tau = |TS(l)|$; /* total number of tasks in level l */
        $TotalInst = \sum_{i=1}^{\tau} \theta(t_i)$; /* total number of instruction */
        foreach *task* t: $TS(l)$ **do**
            $ReqCores = \lceil \frac{\Delta_{core}}{TotalInst} * \theta_t \rceil$;
            $ReqMem = \lceil \frac{\Delta_{mem}}{TotalInst} * \theta_t \rceil$;
            Assign $ReqCores$ and $ReqMem$ amount of resource to the current task t ;
        end
    end
end

Algorithm 1. CPU-intensive scheduling algorithms.

Table 2
Resource distribution among tasks.

Total avail cores		10	
Amount of memory (GB)		32	
Task	#Instruction	AssignCores	AssignMem
t1 **(Medication suggestion)**	1000	2	5.3
t2 **(Digital x-ray image analysis)**	2500	4	13.3
t3 **(Patient report summary)**	700	1	3.8
t4 **(Potential disease prediction)**	1800	3	9.6

The distribution of resources is carried out based on the number of instructions in the respective tasks. Let, $\theta(t_i)$ be the number of instructions in task t_i. The DAG can be treated as a tree with the main task as root of the tree. Level of the DAG is similar to the level in tree, where the root or main task of the tree is numbered as level 0. Let, τ be the variable consisting of a set of tasks at different level. The central idea behind this scheduling algorithm is task having maximum number of instructions must get maximum resources to complete its execution on time and vice-versa. The inner-most loop in the Algorithm 1 is responsible for the distribution of currently available cores and memory among all tasks in the same label. For the sake of clarity, let us consider an example presented in Table 2.

From the above example, it can be observed that fair amount of resources are allocated to all the tasks based on their workload. Since, task t_2 consists of 2500 number of instructions for analyzing the digital image, maximum amount of resources should be assigned to it and therefore 4 cores with 13.3GB of memory are assigned. On the other hand, task t_3, which displays patients past health record consists of less number of instructions and therefore 1 core with 3.8GB of memory are assigned to it. Here, our presumption is that task having maximum number of instructions need more resources to complete in less time. This distribution is made to complete all tasks at the same time so that internal waiting time caused by different completion time of tasks in the same level can be removed. Unlike other schedulers, where estimation of completion time plays an important role in scheduling and resource allocation, here the workload of tasks in a job decides the scheduling and resource allocation. After deciding which task requires how much resource, all tasks are forwarded to different VMs with their required configuration.

5.2. data-intensive job scheduler

Usually data or I/O-intensive jobs are mostly involved in Input/Output operation. These jobs are capable of processing large data sets. Since, data are distributed in different locations, they need to be placed locally to process them. Under this condition, instances will be mostly busy in reading and writing operations. From operating system concept, we assume that a VM may involve in both computation and communication. From this assumption, we can conclude that other independent tasks can be assigned to the processor while an instance is busy in collecting data for execution of a task. This is the central idea of the data-intensive job scheduler, which we have referred to as the work in [6], where a node is assigned with two VMs, one in foreground and other in the background. Background VM runs only when the foreground VM remains idle. In our proposed mechanism, two independent tasks are assigned to the single virtual machine. One is of data-intensive task that runs in foreground and another is either data or CPU-intensive task that runs in the background. The background task starts its execution only when the instance will be busy in large data transfer. The proposed data-intensive scheduler has two stages: primary task assignment for foreground execution, which are discussed as follows.

Stage-1: Algorithm 2 shows the assignment of primary tasks based on BestFit policy. *INS* is the list of all VMs. In Line 3, the list *INS* is sorted in descending order based on the resources available to assign VM. For example, VM *A* configured with 4 cores, 8GB of memory and 1 gbps of network bandwidth must appear before another instance *B* configured with 4 cores, 8GB of memory and 250 mbps of network bandwidth. *T* represents the list of tasks. Based on the workload (number of instructions and amount of data to be processed) *T* will be sorted in descending order as shown in Line 5. *BestInstance* function will return the VM *BI* from the list *INS*, which is suitable for the task *t* as given in Line 7. *BestInstance* can be defined as the virtual machine configured with exact amount of resources required by a task. Scanning and assignment of *BestInstance* will be repeated for each task *t* in the list *T* provided the list *INS* is not empty.

Stage-2: In this stage, utilization of resources is optimized by assigning secondary task to each VM, which will run in the background. Before assigning the secondary tasks, it is essential to estimate the network usage of each VM. This can be determined by the required amount of data to be read from other locations and the bandwidth of the VM. Higher value of network usage indicates larger idle time of the respective processors. A virtual machine having more network usage than the predefined network usage threshold for secondary task is eligible to handle another task as background task.

Instead of assigning a randomly selected secondary task to an eligible VM, the computation workload of a task is calculated and is assigned to the eligible VM. From the set of secondary tasks, the task having maximum computation workload as given in Line 16 is assigned to the instance having maximum network usage value, as given in Line 17 and same is repeated for each secondary task until there is no eligible VM available.

Based on our concept of scheduling the healthcare jobs, we have designed a scheduling framework that consists of two simple but effective scheduling algorithms applicable for different varieties of healthcare jobs. The advantages of both scheduling algorithms are

```
1  INS = List of VMs;
2  T = List of all tasks ;
3  Sort INS based on their available resources (in descending order);
4  while INS is not Empty OR T is not Empty do
5  |  Sort T based on their workload (in descending order);
6  |  t = Task with maximum workload in T;
7  |  t.Assign ← BI ← BestInstance(INS) ;
8  |  Remove task t from T ;
9  |  Remove instance BI from INS;
10 end
   /* Assignment of secondary task                                          */
11 ST =set of secondary tasks ;
12 Prepare the set of VMs INS ;
13 INS = SortOnNetworkUsage(INS) ;
14 while INS is not Empty OR ST is not Empty do
15 |  vm ← TOP(INS);
16 |  t^s = MaxComputionWorkload(ST) ;
17 |  t^s.Assign ← vm;
18 |  Remove vm from INS;
19 |  Remove t^s from ST;
20 end
```

Algorithm 2. data-intensive scheduling algorithm.

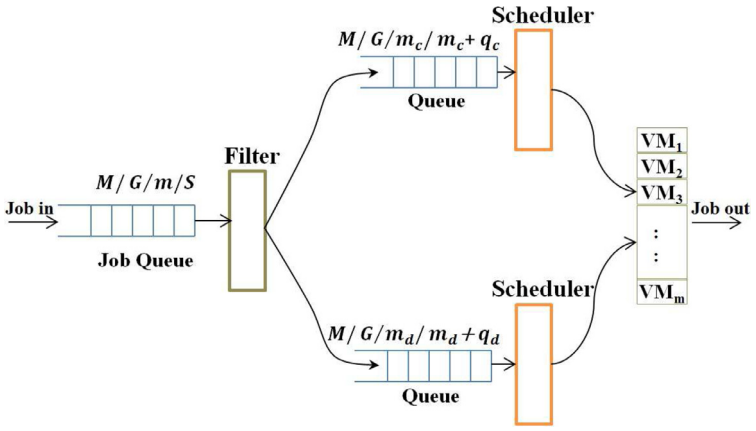


Fig. 4. Analytical model of the proposed framework.

summarized as follow.

- The emergency jobs can be scheduled without any delay by giving the higher preference over other jobs.
- Resources can be utilized efficiently by assigning multiple jobs to the single VM by data-intensive scheduler.
- Waiting time of the secondary tasks can be reduced by assigning to the VM, which is busy in networking operation.
- Fair distribution of the available resources is made among the CPU-intensive jobs.

We have considered the priority as one of the major parameter in order to identify and schedule the emergency jobs which is one of the major advantages. In addition to that, fair distribution of the available resources among the incoming CPU-intensive jobs is one of the major advantages of the CPU-intensive scheduling algorithm.

6. Analytical model

In this section, the proposed mechanism is analyzed and formulated mathematically. The proposed system is modeled as an $M/G/m/S$ queuing model with a finite capacity of job buffer at the top and multiple multi-task jobs arrivals as depicted in Fig. 4. As per discussed mechanism, in Section 4, we have two schedulers each having their own dedicated queue. Hence, the whole system is divided into two sub-systems possessing own queuing model. Let q_c be the length of queue for CPU-intensive scheduler whereas q_d be the length of queue for data or I/O-intensive scheduler, which depends on the number of tasks in the waiting queue. m represents the total number of finite servers. In our case, in this section server refers to virtual machines. Mathematically $m = m_c + m_d$ where $m_c = |VM_c|$ and $m_d = |VM_d|$. Since, it is hard to predict the value of m_c and m_d, we assume that the total number of servers (m) is equally distributed among VM_c and VM_d. Hence m_c and m_d both have same value. S denotes the system capacity, given by,

$$S = (m_c + q_c) + (m_d + q_d). \tag{2}$$

Hence, for both the sub-systems the queuing model can be written as $M/G/m_c/m_c + q_c$ and $M/G/m_d/m_d + q_d$ for CPU and data-intensive jobs, respectively. It is difficult to decide the value of m_c and m_d, as m is assumed to be constant and the value of m_c and m_d can increase or decrease depending on the number of tasks in corresponding queues.

Let $N_x(t)$ be the number tasks currently in the system, which are either in running state or waiting state. x represents CPU-intensive and data-intensive. Let, $\beta_x(t)$ be the number of tasks currently running in the system. By assuming that, one VM can execute utmost one task at a time, the relationship between $\beta_x(t)$ and m_x can be derived as

$$\beta_x(t) < m_x. \tag{3}$$

This indicates that, at any given point of time, the number of tasks running in the system is less than the total number of virtual machine present in the repository. The mean number of tasks in the system can be derived as

$$\overline{N}_x(t) = \sum_{k=0}^{m} k p_k, \tag{4}$$

where p_k represents the probability that the system has k number of tasks. Similarly, the expected number of tasks in the waiting queue can be written as

$$N_{q_x}(t) = \sum_{k=2}^{m} (k-1) p_k. \tag{5}$$

Using Littleś formulas, the mean service time or mean execution time and expected number of tasks in the system can be rewritten

as

$$\overline{T}_{E_x} = (\lambda_x)(\omega_x), \tag{6}$$

$$T_{q_x} = (\lambda_x)(\omega_{q_x}), \tag{7}$$

where λ_x represents the arrival rate of tasks. ω_x and ω_{q_x} are the mean time spent in the system, that includes waiting time and execution time, and mean time spent in the waiting queue, respectively. If the probability distribution is assumed to be Poisson probability distribution, the value of p_k at time t can be written as

$$p_k(t) = \frac{(\lambda_x t)^k}{k!} e^{-\lambda_x t}. \tag{8}$$

Applying above derivation in Eq. (4), the mean number of tasks can be calculated as

$$\begin{aligned} \overline{N}_x(t) &= \sum_{k=0}^{m} \frac{k(\lambda_x t)^k}{k!} e^{-\lambda_x t} \\ &= \sum_{k=0}^{m} \frac{(\lambda_x t)^k}{(k-1)!} e^{-\lambda_x t}. \end{aligned} \tag{9}$$

Similarly, the expected number of tasks in the waiting queue as presented in Eq. (5) can be re-written as

$$N_{q_x}(t) = \sum_{k=2}^{m} \frac{(k-1)(\lambda_x t)^k}{k!} e^{-\lambda_x t}. \tag{10}$$

The derivation in Eqs. (9) and (10) can be used to adjust the value of m_c and m_d. In other words, the equations can be used to keep the required number of virtual machines configured.

Before a multi-task healthcare job gets response from the server, it has to pass through two different queues/buffers. Initially a job has to be in the buffer and wait for the filtration process followed by another queue where actual scheduling policy is being implemented. As discussed earlier in Section 4, filtration should be performed in real-time manner irrespective of the arrival rate and hence the waiting time on the queue can be considered as negligible. Jobs are filtered in first come first serve fashion.

When a healthcare job is submitted to the scheduler, the job breaks into multiple tasks and it is assumed that their service time is identically and independently distributed according to a general distribution. Tasks are then submitted to the respective VMs for execution. Communication among tasks may require for intermediate data exchange. After completion of all tasks, final result can be obtained. Thus the final Result Acquiring Time (RAT), T_{RAT_x} is given by

$$T_{RAT_x} = T_{W_x} + T_{E_x}, \tag{11}$$

where T_{W_x} and T_{E_x} denotes the waiting time and execution time, respectively.

7. Implementation and simulation

This section gives the detail of performance evaluation through real implementation onto our testbed and simulation, which is done using CloudSim simulator. The experimental environment, the results obtained through implementation and simulation are described in following subsections.

7.1. Implementation setup

The physical resources are utilized efficiently by implementing virtualization technology. The virtualization is done in two ways: hypervisor-based virtualization and container-based virtualization. Containers are implemented at operating system level, which gives major advantage in terms of performance and resource consumption. For the evaluation purpose of the proposed scheduling framework, the container-based virtualization provided by Docker platform is used for tasks execution.

Fig. 5 shows the control flow and the components interactions of the implemented framework. All the components are executed simultaneously. The implementation architecture consists of different components such as component for creating the tasks randomly, filtering the tasks, scheduling the filtered tasks, and component for creating and running Docker containers. The tasks are created in the form of files, called task file. Task files are created at random time interval. Each task file contains the information regarding task type, resource requirement, and the command, which needs to be performed by the container. In parallel, another shell script file is executed to move the task files to the corresponding directories. Two directories are created for CPU-intensive tasks and data-intensive tasks. Both the directories represent the queue for each scheduler. The order at which task files need to be scheduled is decided by task file creation time. Inside each directory, *running* directory is created. A task file inside *running* directory represents the task in running state. After the execution of task, the respective task file is deleted from *running* directory.

Similar to the above-mentioned components, both the schedulers are executed concurrently. Each scheduler read the task files from its respective queue directory and move the selected task file to *running* directory, which resides in queue directory. The scheduler also interacts with Docker engine to create and run the containers. The following code snippet is used to interact with Docker engine to launch a new container.

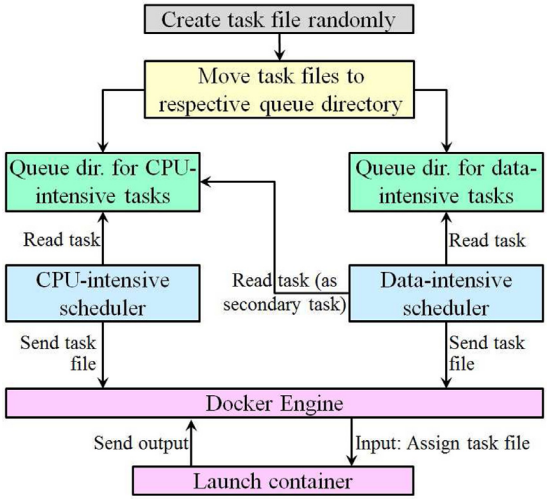


Fig. 5. Implementation architecture of proposed framework.

Containers are removed after the assigned CPU-intensive tasks are finished. For data-intensive task scheduler, following code snippet is used to send both primary and secondary task to the active container.

In order to demonstrate the proposed scheduling framework, we have implemented the CIJS and DIJS algorithms onto our testbed. The testbed is composed of 20 high-end systems: each powered by Intel core i7-4790 @ 3.6GHz CPU, equipped with 8GB memory and 1TB of storage with Ubuntu 14.04 installed on each system.

7.2. Implementation results

Fig. 6 shows the start-up time of virtual machines and containers in Docker platform for each server. Start-up time refers to the time required to load the necessary OS kernel in order to execute the assigned task. It can be observed from Fig. 6 that the startup time increases at a higher rate when the number of virtual machines increases. On the other hand, the startup time for Docker containers is very less as compared to that of virtual machines. The startup time for virtual machines is ranging from 24 sec to 60 sec, whereas the startup time for container is ranging from 0.4 sec to 22 sec. The reason behind the faster start-up time for a container in contrast to that of a VM is the amount of physical resource required to start. Containers in Docker platform shares the same kernel provided as Docker engine, whereas VMs created using hypervisor-based virtualization technology posses the complete OS and the kernel. As a result, the resource requirement to run a VM is more than running a Docker container.

Fig. 7 shows the effect of task type and the type of virtualization environment on tasks' response time. The response time refers to the sum of startup time of the containers/VMs and the time taken for loading the task itself and its supporting library files onto the memory. From the experimental result, it is observed that the response time of the tasks depends on the underlying virtualized environment rather than the type of the task. In both Fig. 7(a) and (b), the average response time of the CPU and data-intensive tasks

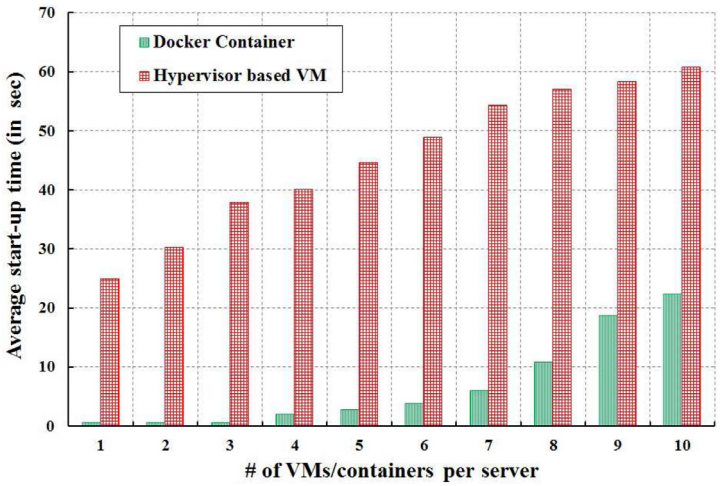
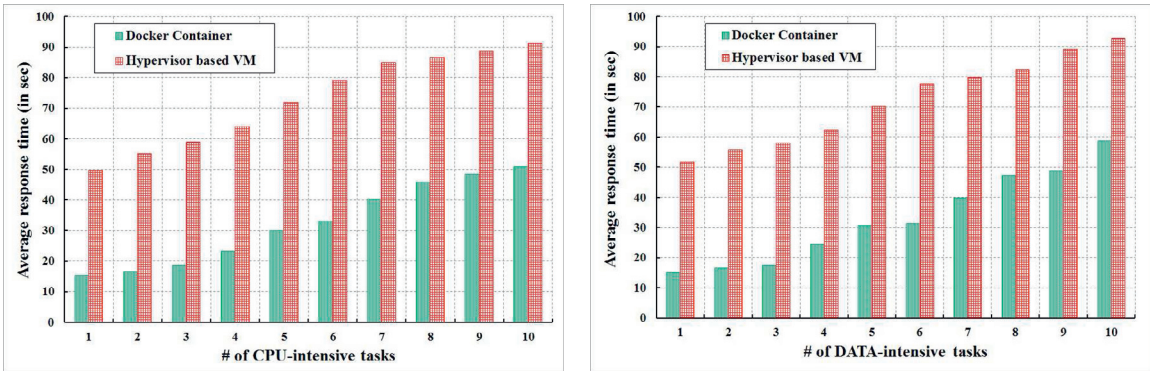


Fig. 6. Start-up time of VMs/Containers



(a) Average response time of CPU-intensive tasks (b) Average response time of data-intensive tasks

Fig. 7. Average response time of CPU and data-intensive tasks.

is less when they are assigned to containers. The average response time for CPU-intensive tasks running in containers ranges from 15 sec to 53 sec, whereas the response time ranges from 49 sec to 91 sec. Similarly, the response time for data-intensive tasks in VMs ranges from 51 sec to 92 sec, whereas the response time of data-intensive tasks in containers ranges approximately from 14 sec to 58 sec.

Experimental result for average execution times of CPU and data-intensive tasks are shown in Fig. 8. The execution time of a task refers to the sum of actual running time and waiting time. In other words, execution time is the time interval between arrival time and finish time, which includes time spent in waiting queue. The type of the task has no effect on its execution time, as the containers are pre-configured with suitable amount of resources. The average execution time ranges from approximately 14 sec to 53 sec when the number of CPU-intensive tasks are increased from 1000 to 10000, whereas the average execution time increases from 15 sec to 63 sec when the same number of data-intensive tasks are executed in containers.

The resource utilization by the CPU-intensive tasks that are scheduled by proposed CIJS algorithm, is shown in Fig. 9(a). In the experiment, the resource utilization includes CPU utilization and the utilization of network bandwidth provided by the container. The utilization of any resource type is calculated as the percentage of the available amount of resources allocated to all the tasks running on the respective container. Presuming that, CPU-intensive tasks require less amount of network resources, the containers are pre-configured with higher amount of CPU resource and less amount of network resource. From the experimental result, in Fig. 9(a), it is observed that, for CPU-intensive tasks the utilization of CPU, which ranges from 60% to 95%, is higher than the utilization of network bandwidth, which ranges from 43% to 69%.

Similar to experimental result on resource utilization shown as in Fig. 9(a), we have also experimented on the resource utilization by data-intensive tasks, as shown in Fig. 9. The proposed DIJS scheduling mechanism in Algorithm 2, schedule the incoming data-intensive tasks. Fig. 9(b) shows the resource utilization, when only data-intensive tasks are assigned to the containers, whereas Fig. 9(c) shows the resource utilization by data-intensive tasks in conjunction with CPU-intensive tasks. It is observed that the network utilization, as well as the CPU utilization, is higher when data-intensive tasks are scheduled to run in a container in conjunction with CPU-intensive task at background. Comparing the Fig. 9, it is observed that the CPU resource utilization, which ranges

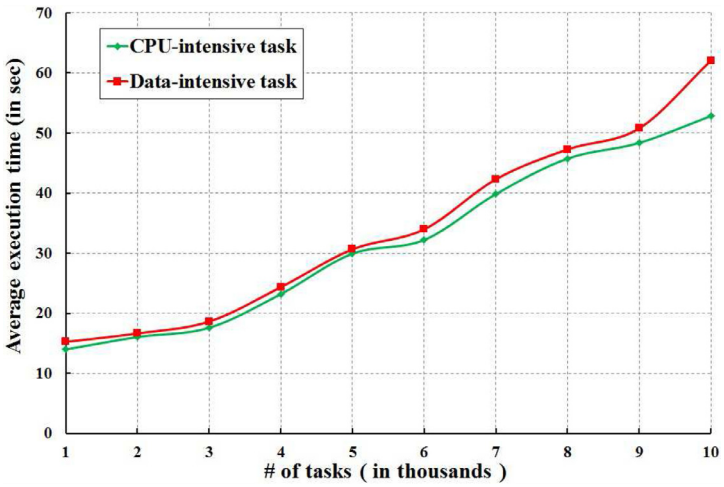
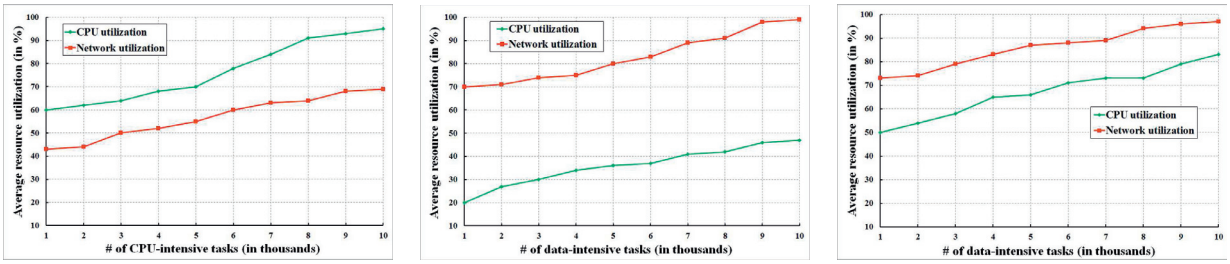


Fig. 8. Average execution time of CPU and data-intensive tasks.



(a) Resource utilization by CPU-intensive tasks (b) Resource utilization by data-intensive tasks (without secondary task) (c) Resource utilization by data-intensive tasks (with secondary task)

Fig. 9. Resource utilization by CPU and data-intensive tasks.

from 20% to 47%, is not efficient and remain idle when the assigned task is busy using network resource. In order to improve the utilization of CPU resource, another CPU-intensive task is assigned to run in background and as a result utilization of CPU boosts up and ranges from 50% to 83%.

7.3. Simulation results

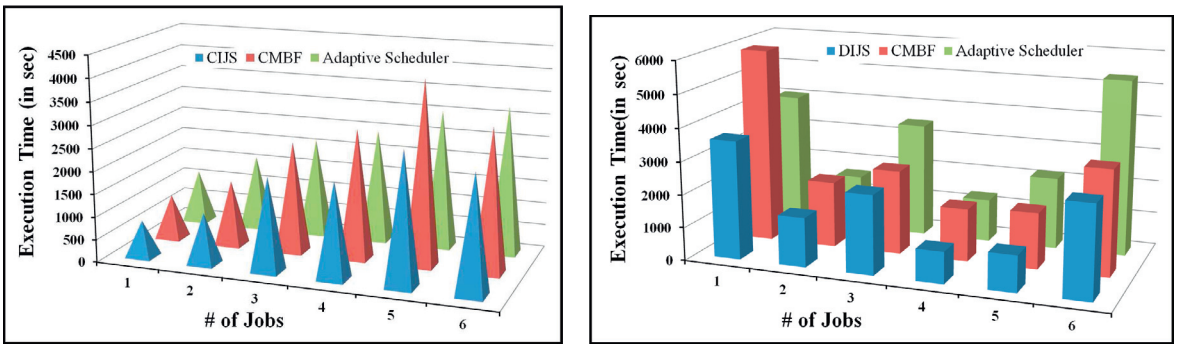
We evaluate the performance of our algorithms using CloudSim simulator, which is popular exclusively for cloud environment. In this section, we present performance results to show the effectiveness of proposed scheduling technique and compare them with CMBF [6] and Adaptive Scheduler [22]. In the following discussion, we have explained the utilization of VMs, response and execution time of both types of jobs. We have demonstrated how dedicated VMs and scheduler for different types of jobs helped us to minimize the makespan of incoming jobs and optimize the utilization of hosted VMs.

The two exclusive algorithms lead to better performance in terms of execution time. For comparison, we have taken 12 jobs with varies number of tasks. Out of them, 6 jobs are data-intensive jobs and same numbers of jobs are of CPU-intensive jobs. As shown in Fig. 10(a), the CPU-intensive Job Scheduler (CIJS) is used to schedule the jobs to simulate the execution time for all 6 CPU-intensive jobs. It is considered that CPU-intensive jobs are loosely coupled tasks and number of tasks are not huge. The tasks are not network-intensive. Hence, the VMs, which are going to host the jobs must have high computing capacity rather than network bandwidth. These specially configured VM results in reduction of execution time of the jobs as shown in Fig. 10(a). Though the elapsed time of job1 is 798 sec, it gradually increases for more number of jobs.

Fig. 10(b) shows the execution time for different number of jobs in data-intensive task execution. It shows significant improvement by our proposed scheduling and resource allocation techniques over others. As compared to the algorithms in [6] and [22], performance in terms of elapsed time can be gained up to 20 through 30 percent. Giving more importance to the network bandwidth rather than higher computing capacity leads to minimize the communication time, which eventually helps to minimize the overall elapsed time of a job.

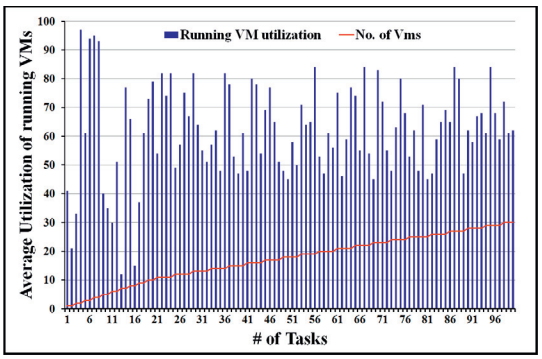
Fig. 11(a) and (b) shows the utilization of different resource types. Fig. 11(a) shows the utilization of VMs with different number of tasks. It shows, with increasing number of tasks the required number of VMs increases. This maintain the trade-off between green computing and system overloading by running exactly the desired number of VMs. It can observe that the utilization of running VMs is approximately consistent. In Fig. 11(a), all the VMs are dedicated to execute CPU-intensive tasks.

On the other hand, Fig. 11(b) shows the utilization of VM including the network bandwidth. For data-intensive tasks, VMs are

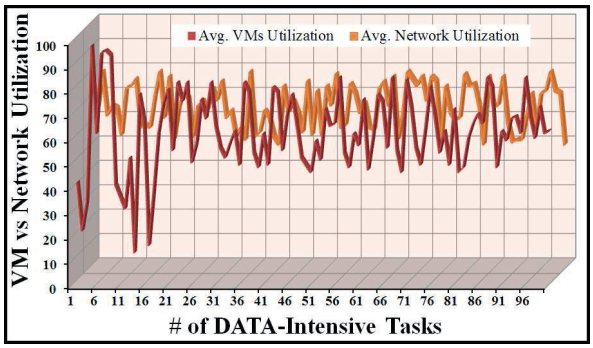


(a) Comparison of CPU-intensive task execution with others (b) Comparison of data-intensive task execution with others

Fig. 10. Comparison of task execution with other algorithms.



(a) Average Utilization of all running VMs



(b) Average Utilization of all running VMs and network bandwidth

Fig. 11. Utilization of VM and network bandwidth.

configured with higher network bandwidth unlike VMs for CPU-intensive tasks. From the simulation result presented in Fig. 11(b), it is observed that consistent utilization of assigned network resource is maintained. Generally, a VM gets busy in I/O operation when a data-intensive task arrives into it. Hence, the utilization of CPU decreases, whereas utilization of network bandwidth increases. The average utilization of network resource of a VM is more during the entire lifetime of a task in that VM. This indicated that most of the time, VM is involved in data transfer rather the execution of the task.

For further more detail evaluation for our proposed scheduling technique, we have presented the graph showing the percentage of VM running (both CPU and data) from the VM repository in Fig. 12. As we discussed earlier that VM repository will be responsible for keeping an inventory of all VMs. In the repository, we kept 60 numbers of VMs for each type of task. Since, the data-intensive tasks require more number of VMs as compare to CPU-intensive task, the percentage of running VM is more in case of VM repository data-intensive tasks. When maximum number of tasks i.e. 100 tasks started execution in parallel the percentage of VM required is 81% in case of CPU-intensive task while it is 90% in case of data-intensive tasks.

Fig. 13(a) shows the average response time of both types of tasks. It is observed that the type of the task has no effect on the response time since both types of tasks were treated separately. The average response time increases when number of tasks increases up to 20, but gradually the rate of increase decreases as number of task increases to 100. Simulation result for average execution time of tasks is shown in Fig. 13(b). It is observed that execution time increases for less number of VM assignments to execute the tasks. However, more number of VM assignments may lead to under-utilization of resources. With the current number of VM assignments to a task, the execution time can be obtained as shown in the figure. It is found that the fluctuation in average execution time is less when the number of tasks increases. From the results as described above, we can summarize that proper configuration of virtual machines and proper assignment of tasks to those virtual machines plays an important role in deciding the overall execution time of a healthcare job, which can be achieved by treating each type of job separately with dedicated scheduling and resource allocation technique.

8. Conclusion

A new scheduling framework is designed to analyze the healthcare applications in cloud, which can work atop cloud management

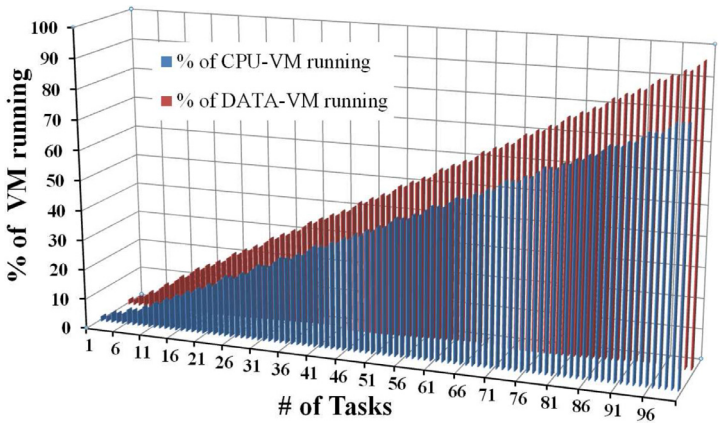
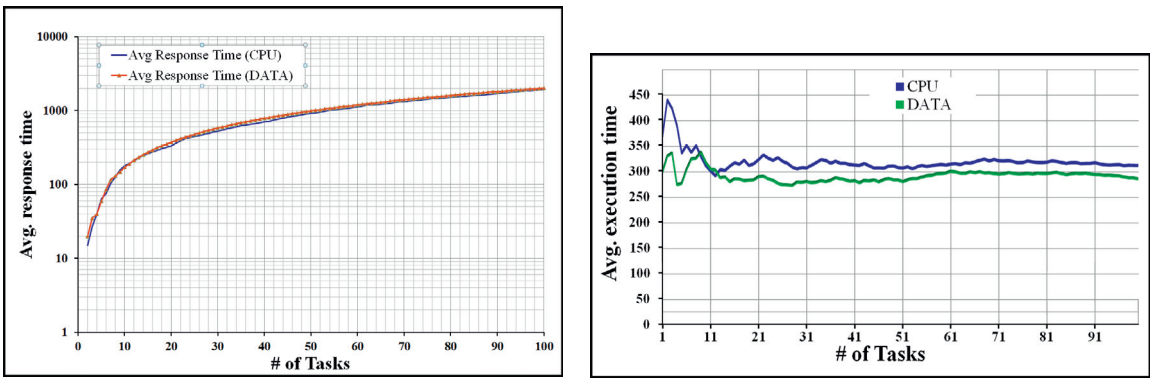


Fig. 12. Comparison of running VMs of both types.



(a) Average response time of both CPU and data-intensive tasks.

(b) Average execution time of CPU and data-intensive tasks.

Fig. 13. Average response and execution time.

solutions. Filtrating process is used in the proposed framework by means of which each healthcare related job can be classified and identified as data or CPU-intensive job. The components of this framework are placed in a loosely coupled manner so that coupling of these components can enable the framework to run in highly parallel manner. In addition to the categorization and scheduling of the healthcare jobs, the proposed framework can also provide a mechanism for the fault detection and recovery. The proposed fault detection and recovery mechanism is implemented by keeping the copies of virtual machine instances in two VM image databases. Our future work involves the optimization of efficiency of filtrating process of our proposed algorithms. By considering the proposed scheduling mechanisms and advantages of the framework, we believe that this framework could be very useful for the cloud data centers to provide fast and fault tolerant healthcare services.

Acknowledgment

This work is supported by Ministry of Science and Technology (MOST), Taiwan under the grant number 106-2221-E-182-014.

References

[1] Li Z, Liao L, Leung H, Li B, Li C. Evaluating the credibility of cloud services. Comput Electr Eng 2017;58:161–75. http://dx.doi.org/10.1016/j.compeleceng.2016.05.014.
[2] Docker Build, ship, and run any app, anywhere. 2017. URL https://www.docker.com/.
[3] Roh H, Jung C, Kim K, Pack S, Lee W. Joint flow and virtual machine placement in hybrid cloud data centers. J Network Comput Appl 2017;85:4–13. http://dx.doi.org/10.1016/j.jnca.2016.12.006.
[4] Cheng D, Rao J, Guo Y, Jiang C, Zhou X. Improving performance of heterogeneous MapReduce clusters with adaptive task tuning. IEEE Trans Parallel Distrib Syst 2017;28(3):774–86. http://dx.doi.org/10.1109/tpds.2016.2594765.
[5] Guo L, Zhao S, Shen S, Jiang C. Task scheduling optimization in cloud computing based on heuristic algorithm. J Networks 2012;7(3):547–53.
[6] Liu X, Wang C, Zhou BB, Chen J, Yang T, Zomaya AY. Priority-based consolidation of parallel workloads in the cloud. Parallel Distrib Syst, IEEE Trans 2013;24(9):1874–83.
[7] Zhou J, Cao Z, Dong X, Xiong N, Vasilakos AV. 4s: a secure and privacy-preserving key management scheme for cloud-assisted wireless body area network in m-healthcare social networks. Inf Sci 2015;314(0):255–76.
[8] Vijayakumar P, Pandiaraja P, Karuppiah M, Deborah LJ. An efficient secure communication for healthcare system using wearable devices. Comput Electr Eng 2017. http://dx.doi.org/10.1016/j.compeleceng.2017.04.014.
[9] Mahmud S, Iqbal R, Doctor F. Cloud enabled data analytics and visualization framework for health-shocks prediction. Future Gener Comput Syst 2016;65:169–81. http://dx.doi.org/10.1016/j.future.2015.10.014.
[10] Sailunaz K, Alhussein M, Shahiduzzaman M, Anowar F, Mamun KAA. CMED: cloud based medical system framework for rural health monitoring in developing countries. Comput Electr Eng 2016;53:469–81. http://dx.doi.org/10.1016/j.compeleceng.2016.02.005.
[11] Jrad F, Tao J, Brandic I, Streit A. Sla enactment for large-scale healthcare workflows on multi-cloud. Future Gener Comput Syst 2015;43:135–48.
[12] Rallapalli S, Gondkar R, Ketavarapu UPK. Impact of processing and analyzing healthcare big data on cloud computing environment by implementing hadoop cluster. Procedia Comput Sci 2016;85:16–22.
[13] Wang Y, Shi W. Budget-driven scheduling algorithms for batches of mapreduce jobs in heterogeneous clouds. Cloud Comput, IEEE Trans 2014;2(3):306–19.
[14] Ran Y, Yang J, Zhang S, Xi H. Dynamic IaaS computing resource provisioning strategy with QoS constraint. IEEE Trans Serv Comput 2017;10(2):190–202. http://dx.doi.org/10.1109/tsc.2015.2464212.
[15] Polverini M, Cianfrani A, Ren S, Vasilakos A. Thermal-aware scheduling of batch jobs in geographically distributed data centers. Cloud Comput, IEEE Trans 2014;2(1):71–84.
[16] Mo L, Kritikakou A, Sentieys O. Decomposed task mapping to maximize QoS in energy-constrained real-time multicores. 35th IEEE international conference on computer design (ICCD). Boston, United States: IEEE; 2017. p. 6.
[17] Chen H, Zhu X, Guo H, Zhu J, Qin X, Wu J. Towards energy-efficient scheduling for real-time tasks under uncertain cloud computing environment. J Syst Software 2015;99:20–35.
[18] Zhou J, Yan J, Wei T, Chen M, Hu XS. Energy-adaptive scheduling of imprecise computation tasks for qos optimization in real-time mpsoc systems. Proceedings of the conference on design, automation & test in Europe. 3001 Leuven, Belgium, Belgium: European Design and Automation Association; 2017. p. 1406–11.
[19] Warneke D, Kao O. Exploiting dynamic resource allocation for efficient parallel data processing in the cloud. Parallel Distrib Syst, IEEE Trans 2011;22(6):985–97.
[20] Liroz-Gistau M, Akbarinia R, Agrawal D, Valduriez P. FP-hadoop: efficient processing of skewed MapReduce jobs. Inf Syst 2016;60:69–84. http://dx.doi.org/10.1016/j.is.2016.03.008.
[21] Hsu C-H, Slagter KD, Chung Y-C. Locality and loading aware virtual machine mapping techniques for optimizing communications in mapreduce applications.

Future Gener Comput Syst 2015;53:43–54.
[22] Polo J, Becerra Y, Carrera D, Steinder M, Whalley I, Torres J, et al. Deadline-based mapreduce workload management. Network Serv Manag, IEEE Trans 2013;10(2):231–44.
[23] Coninck ED, Verbelen T, Vankeirsbilck B, Bohez S, Simoens P, Dhoedt B. Dynamic auto-scaling and scheduling of deadline constrained service workloads on iaas clouds. J Syst Softw 2016;118:101–14.
[24] Kailasam S, Gnanasambandam N, Dharanipragada J, Sharma N. Optimizing ordered throughput using autonomic cloud bursting schedulers. Softw Eng, IEEE Trans 2013;39(11):1564–81.
[25] Thain D, Tannenbaum T, Livny M. Distributed computing in practice: the condor experience. Concurrency Comput 2005;17(2-4):323–56.

Prasan Kumar Sahoo received M.Sc and Ph.D in Mathematics from Utkal University, M.Tech in Computer Science from IIT, Kharagpur, India and 2nd Ph.D in Information Engineering from National Central University, Taiwan. He is Professor in Information Engineering department, Chang Gung University, Taiwan, Researcher in Division of Colon and Rectal Surgery and Associate Researcher in department of Cardiology, CGMH, Linkou, Taiwan.

Chinmaya Kumar Dehury received BCA degree from Sambalpur University, India, in June 2009 and MCA degree from Biju Pattnaik University of Technology, India, in June 2013. Currently he is pursuing the PhD in the department of Computer Science and Information Engineering, Chang Gung University, Taiwan. His research interests are in scheduling, resource management and fault tolerance problems of Cloud Computing.