Computing and Informatics, Vol. 40, 2021, 104–144, doi: 10.31577/cai_2021_1_104

MULTI-OBJECTIVE TASK SCHEDULING USING SMART MPI-BASED CLOUD RESOURCES

Mehran Mokhtari

Department of Computer, Sari Branch, Islamic Azad University, Sari, Iran
e-mail: mehrmokhtari@yahoo.com

Peyman Bayat*

Department of Computer, Rasht Branch, Islamic Azad University, Rasht, Iran
e-mail: bayat@iaurasht.ac.ir

Homayun Motameni

Department of Computer, Sari Branch, Islamic Azad University, Sari, Iran
e-mail: h_motameni@yahoo.com

Abstract. Task Scheduling and Resource Allocation (TSRA) is the key focus of cloud computing. This paper utilizes Smart Message Passing Interface based Approach (SMPIA) and the Roulette Wheel selection method in order to determine the best Alternative Virtual Machine (AVM). To do so, the Virtual MPI Bus (VMPIB) is employed for efficient communication among Virtual Machines (VMs) using SMPIA. In this matter, SMPIA is applied on different resource allocation and task scheduling strategies. MakeSpan (MS) was chosen as an optimization factor and solutions with minimum MS value as the best task mapping performance and reduced cloud consumption. The simulation is conducted using MATLAB. The analysis proves that applying SMPIA reduced the Total Execution Time (TET) of resource allocation, maximum MS time, and increase the Resource Utilization (RU), as compared to non-SMPIA for Greedy, Max-Min, Min-Min algorithms. It is observed that SMPIA can outperform non-SMPIA. The effect of SMPIA is more

* Corresponding author

obvious as change in the MS and the number of cloud workloads increase. Furthermore, regarding the TET and MS of the tasks, the SMPIA can significantly reduce the starvation problem as well as the lack of sufficient resources. In addition, this approach improves the system's performance more than the previous methods, what reflects effectiveness of the proposed approach concerning the Message Passing Interface (MPI) communication time in the network virtualization. The mentioned text mining work was prepared concurrently after practical evaluation.

Keywords: Cloud computing, SMPIA, TSRA, resource allocation scheduling, roulette wheel, text mining, AVM, starvation

1 INTRODUCTION

Cloud computing is well known as a model that aims at providing resources and services through a network. In this matter, the Task Scheduling and Resource Allocation (TSRA) present the key focus of cloud computing [1]. Message Passing Interface (MPI) is a standard communication protocol, which has become a legal standard for communication between processes and implements a parallel programming using the MPI [2, 3]. Typically, High-Performance Computing (HPC) applications employ the MPI communication [4, 18]. Here, it is worthwhile to mention that as the mapping tasks problem onto resources (workflow tasks scheduling) is known as NP-complete in the cloud computing, several scheduling algorithms have been developed to solve it [5, 13, 14, 15, 16, 29, 38]. The main objective of the workflow scheduling problem is to reduce Total Execution Time (TET) as well as to generate a balance between the resources consumption and the Quality of Service (QoS) [8, 9].

This paper proposes a Smart MPI Approach (SMPIA) to improve MPI communication time and the lack of sufficient resources, as well as to reduce the resource involvement level and starvation problem (large waiting time) of the tasks. Therefore, this paper addresses two issues:

1. whether it is possible to reduce latency of MPI communication time and starvation problem by reducing the average TET and completion time, and
2. whether it is possible to reduce the resource consumption time and involvement level while ensuring efficiency.

To solve the first problem, the SMPIA and the Roulette Wheel selection method are utilized to determine the probability of choosing Alternative Virtual Machines (AVMs). In addition, several AVMs are employed instead of one Virtual Machine (VM) in sub-networks inter-connected. To answer the second question, a suitable model is developed to calculate the Resource Utilization (RU) in order to decrease the level of resource involvement in the cloud and resource consumption time in

such a way that its inputs were MakeSpan (MS) parameters that improved RU and resource consumption time.

To accomplish this aforementioned aim, several experiments are implemented on different TSRA strategies using SMPIA in the Ministry of Communication and Information Technology of Iran (MCITI) dataset. After that, the performance metrics including MS, TET, and RU are measured. The obtained results of the experiment indicate that the SMPIA outperform standard algorithms including Min–Min algorithm [34] in terms of MS. Moreover, the SMPIA performs better than the algorithm developed in [4, 6, 20, 30, 35, 17, 31, 33, 36, 37], in terms of ET, MS and RU, respectively.

This paper aims to allocate the appropriate AVMs collection based on the minimum MS time and optimal mapping of current flow onto the selected AVM. Through choosing the appropriate AVMs and mapping the flow onto them in the shortest MS time in Virtual MPI Bus (VMPIB), this approach can improve MPI communication time, starvation problem and performance in the cloud computing. To do so, this approach encompasses three phases of calculation of resource-ranking, resource selection, and optimal task-resource mapping. In the following, the main contribution and motivation of this paper is described.

1.1 Contribution and Motivation

The main contributions of this paper are as follows:

1. This paper develops a novel method and technique for efficient communication between VMs in MPI-based cloud resources using VMPIB. The key idea is the phases of the resource-ranking, AVM selection, and mapping of the current flow onto the selected AVM. In this way, the probability of choosing an AVM for workflows was determined using the Roulette Wheel selection method.
2. This paper improves the resource consumption time and task scheduling in the cloud computing. In this regard, the parameters of load, capacity, and amount of computed load, Execution Speed (ES) and execution time of each flow on the VM are employed to calculate the minimum MS. Besides, the parameters CPU processing speed of each flow on the VM are considered to calculate the execution time and TET. It is worth noting that this approach is different from the previous conducted approaches because the effect of MPIA is more obvious as changes increase in the MS and the number of cloud workloads.
3. This paper enhances RU, reduction resource consumption, optimal task-resource mapping model in the cloud computing. To accomplish the aim, the memory capacity parameters, memory capacity used, total processing capacity, and processed capacity on the VMs, are exploited to calculate the RU. It is worthwhile to mention that this approach is different from the former reviewed approaches, because decreasing the RU level using heuristic (Max-Min, Min-Min) and Greedy algorithm indicates that mapping the flows onto the appropriate AVM is carried out properly. Analysis of the results demonstrates that the proposed approach

enhances performance in terms of MS up to 55.94 %, while it is up to 55.59 % in terms of the TET based on which RU and involvement level are enhanced up to 12.80 %. The need to conduct this research is to delay MPI communication and starvation problem in the VMPIB. It should be mentioned that utilizing SMPIA on a telecommunications transaction application increases its efficiency.

The motivation for this research is the implementation of text mining on the telecommunications transactions application in order to reduce MPI processing time and solving starvation problem of the tasks. The main novelty of this research is to implement text mining on the telecommunications transaction application in order to achieve proper processing time as well as to manage the proposed cloud system.

The remainder of this paper is organized as follows: the related works are presented in Section 2. A case study is described in Section 3. Process smart MPIA and job migration in the text mining is found in Section 4. Solving the resource allocation scheduling problem using SMPIA is provided in Section 5. Solving the starvation problem using SMPIA is performed in Section 6. Evaluation of SMPIA is given in Section 7. Discussions and analysis are in Section 8. Ultimately, conclusions and future research are presented in Section 9.

2 RELATED WORKS

In this section, some studies conducted on the scheduling and method of resource allocation in the cloud are presented for the optimal resource use.

The papers of [2, 4, 6, 7, 10, 11, 12, 19] confirmed that the implementation of MPI applications is appropriate on the cloud. In [17], the ranking of each task in Heterogeneous Earliest Finish Time (HEFT) algorithm was performed based on the average of task connections and cost of computing between the current task and its substitution. In [20], the Eager Map algorithm for solving mapping problems was proposed in cluster nodes and cores, which was based on a Greedy heuristic in order to match application communication patterns to hardware hierarchies. A novel dynamic task scheduling algorithm was developed in [26] based on an improved genetic algorithm. Then, some experimental results indicate that the proposed algorithm could effectively improve throughput of the cloud computing systems so that it could significantly reduce the execution time of task scheduling. In [27], a novel task-scheduling algorithm termed as Genetic Algorithm-based Customer-Conscious Resource Allocation and Task Scheduling (GACCRATS) is proposed for the heterogeneous multi-cloud environment in order to cope with the gap between frequently changing customer requirement and available infrastructure for the services. After that, the simulation results were compared with the existing scheduling algorithm. The aim was to task-resource mapping of the multi-cloud federation in order to achieve minimum MS time and maximum customer satisfaction. In [28], the problem of allocating Data Center (DC) resources is considered for the cloud enterprise customers who required the guaranteed services on demand. For the higher traffic situation, the heuristic approach was much more suitable, which was analyzed

and then the results are presented for up to 3,200 servers. The proposed heuristic was fast to solve large-scale problems where the Mixed-Integer Linear Programming (MILP) problem was difficult to solve. They developed a novel MILP model as well as alternately a heuristic that was solved in this framework at each review point. In other words, more frequency options for a server mean higher reduction in the energy consumption. According to findings of [28], there are several future directions to address, which do not allow partial fulfillment of a request if there is a lack of sufficient resources to consider fully a request. Furthermore, they planned to add performance evaluation on the loads to a DC based on its geographical distance from different Virtual Networks (VNs). Moreover, they planned to explore different allocation policies so that the service performance was comparable for different VN customer groups. In [32], Foundations of Machine Learning (FOML) algorithm is compared to Min–Min, Max–Min, Sufferage and Enhancement HEFT (E-HEFT) algorithm. In [33], the simulation results show that the Segmented Min-Min (SMM) algorithm with the high number of tasks and machines is the best. In [34], the obtained results indicated that the Max-Min Scheduling Improved Algorithm (MM-SIA) had the lowest completion time of all VMs, as compared to three algorithms such as Max-Min, Min-Min, and Round Robin. In [35], an approach presented called Optimized Process Placement (OPP) found the best placement scheme comparing to all collective communications on all message sizes.

2.1 Comparison the SMPIA with Benchmarks, Algorithms and Methods

In [16], compared to the previous methods, the heuristic method could improve the task response time and resource allocation up to 50 %. The proposed heuristic approach performed task scheduling and resource allocation efficiently with high utility. In this way, the maximum RU was achieved with computing resources such as CPU, memory and bandwidth. Existing systems such as [16] considered three resources of CPU, memory, and bandwith in evaluating their performance. In our proposed system, the parameters such as (CPU, memory, bandwith, storage), capacity (memory, CPU), load (VMs, flow), number (VMs, flows), ES (flow, CPU), DC ID, server ID, CPU number, CPU Freq (HZ), VM ID, flow ID were considered as input parameters to calculate VM capacity. Also, the experimental results show that the SMPIA outperforms three standard algorithms, i.e., Greedy, Min–Min and Max–Min algorithm and improved these algorithms in terms of TET, MS, and RU metrics. The obtained result indicated the lower of the starvation problem between tasks and resources, the lower the level of resource involvement and resource consumption in MPI communications. Besides, the related performance parameters in SMPIA were calculated based on the mean values, as obtained after 50 times of the program execution. This paper conforms results [1, 8, 9, 10, 14, 16, 22, 23, 24, 25, 30, 31, 36] avoiding Service Level Agreements (SLA) violation and QoS dropping.

In this paper, the SMPIA could improve the resource allocation time and completion time up to 55.80% and 55.94 %, respectively, which caused reducing the resource consumption and resource involvement levels up to 11.80 %. At the mean-

time, the RU parameter confirms the rate of resource consumption and the extent of the involvement of hardware resources in the cloud to percentages. In this paper, both high and low percentages do not indicate its high or low quality. Table 1 reports the details of comparison performance parameters the SMPIA with benchmarks, algorithms and methods for optimization.

3 CASE STUDY

In this study, the general model of the telecommunication cloud system was designed. In this model, as illustrated in Figure 1, the VMPIB and the cloud management center were considered. To define the concept of the VMPIB, we considered a topology associated with the connected graph $G = (D, V)$, where $D = \{DC_1, DC_2, DC_3, \ldots, DC_d\}$ and $V = \{VM_1, VM_2, VM_3, \ldots, VM_m\}$. In this matter, we bring the following assumptions to investigate which resource can be allocated to the flow. The function $\varphi : V \to D$ was considered to control the dependencies of flows and tasks of all flows including the set $F = \{F_1, F_2, F_3, \ldots, F_f\}$. In addition, the function $\theta : F \to V$ and $T = \{T_1, T_2, T_3, \ldots, T_t\}$ is regarded as well. Here, D is the total number of DC; V refers to the total number of VMs; T denotes the set of t transaction, it was defined as the total of transactions. F means the total of the flows depending on each other. θ is a function to determine which flow could be executed by VM in Virtuall MPI Bus. In addition, φ is function, in which VM is assigned to each DC in the Virtuall MPI Bus. To do so, we bring the following assumptions:

Definition 1 (Cloud management center)**.** The cloud management center contains the cloud manager (Administrator), the workflow progress manager, and the initial scheduler. It performs the schedule work, schedule workflows, and resources, and then sets the initial values. The job scheduler schedules the workflows and resources, and then adjusts the initial values (Figure 1).

Definition 2 (Cloud provider)**.** The cloud provider was composed from DC and servers, in which each DC had a number of servers, each of which had several VMs in the VMPIB (Figure 1).

Definition 3 (Cloud VMs)**.** A set of VMs, they receive and process the superclouds as the resources. In this work, each VM has an ID and capacity. We have a list of VMs and IDs for the VMs. The VMs in a VMPIB include two-way communication with each other and a server. It should be noted that each VM could process only one type of flow in large numbers. Each DC, server, and VM has an ID in the cloud.

Definition 4 (Job)**.** A transaction involves a number of jobs. In this way, the flows in a VMPIB must pass through a number of jobs to perform a transaction. Each input and output flow was exhibited as an arrow, whereas each job was depicted as a red, yellow and blue circle in Figures 2 and 3. Similar to array cells, a number of jobs generate a task.

Problem	Technique	Platform	Technology	Reference	Metrics	Improvement
Performance	PingPong TCP	CPU, RAM, VM	MPI	[2]	Latency	30 %
Performance	PingPong Open-MX	CPU, RAM, VM	MPI	[2]	Latency	36 %
Performance	Alltoall	CPU, RAM, VM	MPI	[2]	Latency	35 %
Performance	Alltoallv	CPU, RAM, VM	MPI	[2]	Latency	35 %
Performance	HEAT	CPU, RAM, VM	MPI	[2]	Elapse time	30 %
Performance	HEAT	CPU, RAM, VM	MPI	[2]	Latency	30 %
Performance	Allgather	CPU, RAM, VM	MPI	[2]	Latency	33 %
Performance	Allgatherv	CPU, RAM, VM	MPI	[2]	Latency	31 %
Performance	Reduce	CPU, RAM, VM	MPI	[2]	Latency	35 %
Performance	Reduce-Scatter	CPU, RAM, VM	MPI	[2]	Latency	32 %
Performance	Exchange	CPU, RAM, VM	MPI	[2]	Latency	45 %
Performance	Sendrecv	CPU, RAM, VM	MPI	[2]	Latency	35 %
Performance	LAMPICS, MPAR	CPU, RAM, VM	MPI	[3]	Latency	26.5 %
Performance	MCM	CPU	MPI	[4]	ET	30 %
Performance	MCM_CG Class B	CPU	MPI	[4]	ET	16.2 %
Performance	MCM_CG Class C	CPU	MPI	[4]	ET	12.7 %
Performance	NPA	CPU, RAM, Bandwidth	Simulations	[6]	ET	28.3 %
Performance	NPA	CPU, RAM, Bandwidth	MPICH2 on AEC2	[6]	ET	25.4 %
Performance	N-body	CPU, RAM, Bandwidth	MPI	[6]	Performance	41.6 %
Performance	CG	CPU, RAM, Bandwidth	MPI	[6]	Performance	14.3 %
Performance	CMPI	CPU, RAM, Bandwidth	MPI	[6]	TET	14.3 %
Performance	CMPI	CPU, RAM, Bandwidth	MPI	[6]	NCT	33.2 %
Performance	MCM, NAS-CG CLASS B	CPU, RAM, VM	OpenMPI, MPI	[9]	Latency	29.26 %
Performance	NPB	CPU, RAM, VM	MPI	[12]	ET	20 %
TSRA	Heuristic, BATS+BAR	CPU, RAM, Bandwidth, VM	CCS	[16]	Response time	50 %

Continue in next pages

Table 1. Comparison performance parameters of SMPIA with benchmarks, algorithms and methods to optimization

TMLB	LB*	VM	CCS	[17]	MS	15 %
TMLB	Eager Map HPC	CPU, RAM	OpenMPI, MPI	[20]	ET	10.3 %
TSRA	IGATS	CPU, RAM, Bandwidth, Server, DC	CCS	[26]	ET, Response time	Effective improvement
TSRA	GA, TLBO, GACCRATS, COTS	CPU, RAM, DC, Server	CCS	[27]	MS, customer satisfaction	MCS, Minimize MS
Resource allocation	Heuristic approach, MILP	CPU, Server	CPU, Bandwidth	[28]	Energy consumption	Maximum Reduce energy consumption
Performance, load balancing	Min-Max	CPU	CCS	[30]	TET	9 %
Performance, load balancing	Min-Max	CPU	CCS	[30]	TET	7 %
Performance, load balancing	Min-Max	CPU	CCS	[30]	ART	9 %
Performance, Task scheduling	PA-MMSIA	Tasks/Virtual resources	CCS	[31]	ACT	20 %
Performance, Task scheduling	PA-LBIMM	Tasks/Virtual resources	CCS	[31]	MS	13.2 %
Performance, Task scheduling	PA-LBIMM	Tasks/Virtual resources	CCS	[31]	ARU	1.16 %
Load balancing, Energy consumption	FOML	CPU, RAM, VM	Cloud computing,	[32]	ACT	16 %
Performance, Scheduling	SMM	VM	CCS	[33]	MS	6.8 %
Load balancing	LBIMM	CPU, RAM, VM	CCS	[34]	MS	12.5 %
Load balancing	Min-Min	CPU, RAM, VM	CCS	[34]	MS	19.625 %
Performance	OPP_cyclic	CPU, RAM	MPI	[35]	ET	53.6 %

Continue in next page

Performance	OPP_block	CPU, RAM	MPI	[35]	ET	20.4 %
WSRA	OWM	CPU, VM	Grid environments	[36, 37]	Average MS	26 %
Task scheduling	MCC, MEMAX, CMMN	CPU, RAM	CCS	[38]	MS, ACU	MS, cloud utilization
TSRA	Greedy-SMPIA	CPU, RAM, VM, Bandwidth, DC, Server	CCS, SMPIA	This paper	TET	51.10 %
TSRA	Greedy-SMPIA	CPU, RAM, VM, Bandwidth, DC, Server	CCS, SMPIA	This paper	MS	38.84 %
TSRA	Greedy-SMPIA	CPU, RAM, VM, Bandwidth, DC, Server	CCS, SMPIA,	This paper	RU	12.28 %
TSRA	Max-Min-SMPIA	CPU, RAM, VM, Bandwidth, DC, Server	CCS, SMPIA,	This paper	TET	49.91 %
TSRA	Max-Min-SMPIA	CPU, RAM, VM, Bandwidth, DC, Server	CCS, SMPIA,	This paper	MS	55.94 %
TSRA	Max-Min-SMPIA	CPU, RAM, VM, Bandwidth, DC, Server	CCS, SMPIA,	This paper	RU	11.80 %
TSRA	Min-Min-SMPIA	CPU, RAM, VM, Bandwidth, DC, Server	CCS, SMPIA	This paper	TET	55.80 %
TSRA	Min-Min-SMPIA	CPU, RAM, VM, Bandwidth, DC, Server	CCS, SMPIA	This paper	MS	53.04 %
TSRA	Min-Min-SMPIA	CPU, RAM, VM, Bandwidth, DC, Server	CCS, SMPIA	This paper	RU	11.86 %

Definition 5 (Job migration)**.** Job transfer for running from one AVM to another one (Table 3, Figures 4 and 5).

Definition 6 (Task)**.** In this paper, the blue circles form a task in Figures 2 and 3. For example, in Figure 3, flow number 8 is for a task called a tender, which is divided into general deal (i.e. flow 11) and limited deal (i.e. flow 12).

Definition 7 (Transactions)**.** Transactions (including deals and tenders) are categorized into two categories of small and big. A transaction involves a number of jobs. To perform a transaction, the flows in a VMPIB must go through a number of jobs. Each input and output flow was shown as an arrow, in which each job is depicted as a circle. Besides, the big and small transactions (i.e. deals and tenders) were shown as small and big workflows. The related details are illustrated in Figures 2 and 3. Small and big transactions were composed as 28 and 14 jobs, respectively.

Definition 8 (MPI table)**.** MPI table is the same as Virtual MPI page table. Were these pages are placed in physical memory is determined by the page table. An address obtained with the ampersand operator in program language is not a physical address, but a virtual address. Its initial values are set using the scheduling algorithms at the start of the algorithm through the specified data tables, which are updated during program execution and its values change (Figure 8).

Definition 9 (Distributed MPI table)**.** The type of the network is MPI. Because the servers process in parallel in the MPI network that are distributed in the network. The memory allocated to MPI tables is distributed so that the pages are stored in the buffer of the physical memory.

Definition 10 (MPI's flows)**.** Each transaction (deals and tenders) that enters the cloud system contains a number of flows. The transaction flows (deals and tenders) that are exchanged (sending or receiving) between the VMs for processing are called MPI's flows in the VMPIB. In this regard, some application developers encounter the transmissibility problem in communication networks, which led to the definition of a standard for messaging, so-called the MPI. MPI is a standard interface that is independent from hardware, platform, and message-based for parallel applications. Although the MPI sub-layer can be a proprietary protocol, it does not see the protocols of some applications, except MPI.

Thus, MPI is a middle ware and a simple interface. In MPI, it is assumed that communication takes place between a specific group of processes, in which each group contains an ID. On the other hand, each process contains a local ID in each group. The ID group and ID process of either source or destination identify a message uniquely which is utilized instead of the transfer layer address. In this paper, we have a list of MPI flows and their IDs. Each flow contains the amount of load, ES, and execution time on the VM. As illustrated in Figure 2, the MPI's flows of referrals to the supplier dataset (for providing the qualified supplier) and the review

of technical non-approval (for receiving new requests) are designed as circular while other flows are linearly defined. Note that the small and big transactions were 28 and 14 jobs composed, respectively.

Definition 11 (Input parameters)**.** It includes flow (ID, load, speed, number), CPU (ID, number, speed Freq (HZ), DC (ID), server (ID)), bandwidth (Mb/s), VM (ID, load, size, number) and flow (ID, load, number) (Figure 4).

Definition 12 (Output parameters)**.** It includes capacity (bit/Byte) metrics of VMs, current flow and VM with minimum Computational Cost (CC), next flow and VM with low load variance (Figure 4).

Definition 13 (Execution time)**.** Execution time can be modeled as follows:

1. The amount of computational load of the current flow (L_{CF});
2. The speed of CPU execution (ES).

It is the same as TET and performance metric parameter.

Definition 14 (Completion time)**.** The completion time can be modeled as follows:

1. Time spent by execution flow_k of transaction_j on VM_i namely T_{ijk}.
2. The parameter determining the CC of flow_k on VM_i namely CC_i.
3. The parameter ES_i determines the ES of CPU on VM_i.

Definition 15 (Efficiency)**.** Efficiency can be modeled as RU and performance metric.

Definition 16 (Task scheduling)**.** Allocating the flow to the selected VM in the shortest time. Each task is a transaction that consists of a number of jobs. For example, the task scheduling for a tender in Figure 3 means two general and limited tenders, which each run with the VM in the shortest time, in which each workflow consists of a number of tasks while each task consists of a number of jobs and flows.

Definition 17 (Resource allocation)**.** The best AVM is allocated to the current flow.

Definition 18 (Workflow)**.** For example, the tender task of three jobs and two flows was illustrated in Figure 3. Each workflow is a transaction. The cloud input includes two workflows, big and small deals. Here, in order to employ each workflow to be considered as a service in the process of purchasing deals and tenders, a comprehensive and centralized telecommunication supply chain system is introduced. The aims of designing this application are as follows: cost reduction, decreasing administrative bureaucracy, time productivity, accuracy in performing works, integration and focus on the field of supply, reporting system and preparing the management dashboard, the mechanized management of stakeholders and

suppliers, and improving communication and coordination process. This practical application includes all requirements of the supply chain such as the process of ordering and purchasing of inquiries and tenders, service contracts, and communication with the personnel system, etc. To design this practical application, Key Performance Indicators (KPIs) were utilized to decrease the amount of stagnant items in warehouses, to decrease the cycle time of work processes, percentage of centralized purchases, and identifying the products required by the regions to save on the purchases. Many distribution systems and applications are developed on the simple message model provided by the transmission layer. In this simulation, 31 telecommunication regions distributed in 31 provinces were chosen, each of which participated in the tender process of purchasing telecommunication equipment. MPI in the cloud was utilized to improve the tender processing time, to reduce the process transfer delays, to allocate resources to customers at the suitable time, to improve execution time and completion time. The MPI communications are among the VMs, servers, and DCs. The objective of MPI communications is to get the cloud out of the centralized management. This method, in addition to reducing execution time, completion time, the level of engagement, and resource consumption, has also improved the productivity. In this matter, supplying the resources by allocating resources at the right time has also increased productivity.

Definition 19 (WaaS)**.** Workflow as a Service (WaaS) is an emerging concept that offers workflow execution as a service to the scientific community. Note that WaaS is categorized as either Platform as a Service (PaaS) or Software as a Service (SaaS) on the cloud stack service model. With the emergence of WaaS in the cloud, it is more challenging to predict workflow scheduling and estimate the runtime of tasks. In this way, processing a large volume of data needs to predict real-time changes for the resource performance [21].

Definition 20. The comprehensive design and implementation of a comprehensive and centralized supply chain. In order to decrease the costs and administrative bureaucracy as well as to obtain the productivity on time, punctuality, integration and focus in the field of procurement, reporting system and management dashboard, mechanized management of stakeholders and suppliers, improving the communication process and coordination, an application supply chain (including the comprehensive design and implementation of a comprehensive and centralized supply chain) was introduced. This application encompasses all the requirements of the supply chain such as the process of ordering and purchasing inquiries and tenders, service contracts, and communication with the personnel system, etc. Some KPIs were employed to design this application in order to decrease the amount of stagnant items in the warehouses, the time of the work process cycle, the percentage of centralized purchases, as well as to identify the goods required by the regions to save on purchases.

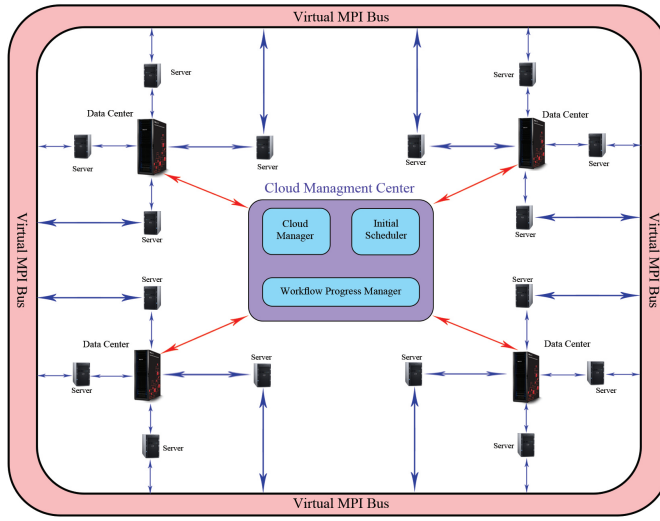

Figure 1. Model of cloud computing based on VMPIB

3.1 Problem Description

In the most basic cloud service model, an infrastructure is considered as a service, in which computing resources can be provided as VMs [2]. The main problem in our work is to allocate suitable AVMs and then optimally map the flows onto them in the minimum processing time in order to

1. simultaneously optimize the performance parameters;
2. to solve the TSRA problem in cloud computing using SMPIA;
3. to address the starvation problem of the tasks (large waiting times for small and big jobs) and the lake of sufficient resources.

To this end, the non-SMPIA and the SMPIA are developed. In the non-SMPIA (Greedy, Max-Min, Min-Min), first, the current flow was chosen based on these aforementioned algorithms from the expected flows. Afterwards, the selected flow was sent to the selected resource of these algorithms. Then, the SMPIA was applied to each of these algorithms in order to optimize TET, completion time, and RU. After optimization, if the VM could not perform the current flow (e.g. the queue was full, the system crashed, the system was disconnected, etc.), the SMPIA was applied. The SMPIA was implemented in three phases: calculation of the AVM ranking, selection of the AVM, and flow mapping onto the AVM. Regarding the MPI management of the VMs, the rank of AVMs was calculated based on the number of its connections with other AVMs based on MS's calculation.

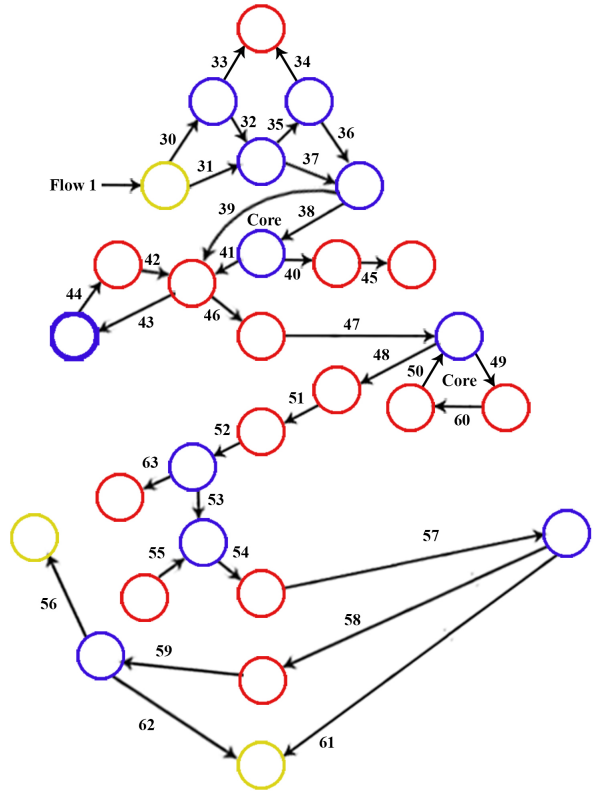

Figure 2. WaaS of small transaction

To calculate the performance parameters, Equations (1), (2), (3), (4) were employed for TET, MS, RU, and average utilization, respectively. On the other hand, the programs were executed at least 50 times in a system to compute the average of the parameters, with the identical specifications. At the end, the mean values were recorded. Regarding the above-mentioned information, all the equations are quickly solved as well.

The execution time is equal to TET using Equation (1):

$$\text{Problem ET} = \frac{\text{The value of calculation load}}{\text{CPU execution speed}} = \frac{L_{cpu}}{ES_{cpu}}. \tag{1}$$

The MS is equal to maximum MS of all tasks using Equation (2):

$$\text{Problem MS} = \max\left(\sum_{i=1}^{m}\sum_{j=1}^{t}\sum_{k=1}^{f}\left(\frac{(CC)_i}{(ES)_i}\right) \times T_{ijk}\right). \tag{2}$$

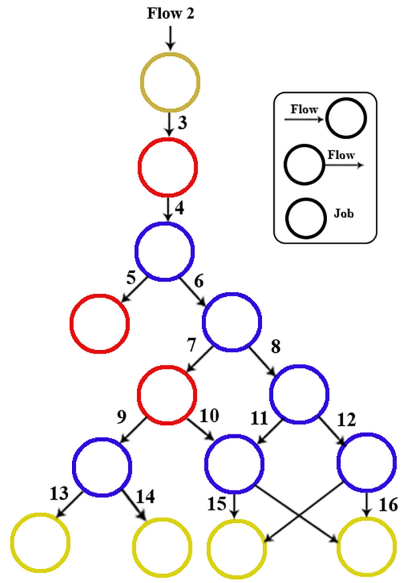

Figure 3. WaaS of big transaction

T_{ijk} is time spent by execution flow_k of transaction_j on VM_i, otherwise, it is zero [22]. CC_i is the parameter determining the CC of flow_k on VM_i. ES_i is the parameter determining the ES of flow_k on VM_i.

The RU is calculated through the following Equation (3):

$$\text{Problem RU}_i = \frac{CP_i}{TPC_i} + \frac{MC_i - MCU_i}{MC_i}. \tag{3}$$

CP_i is the capacity processed in VM_i, TPC_i refers to the total processing capacity of VM_i, MC_i denotes the memory capacity of VM_i, MCU_i means the memory capacity used in VM_i

$$\text{Ave RU} = \frac{\sum_{i=1}^{m} RU_i}{m} * 100. \tag{4}$$

We considered the following hypotheses using Equations (5), (6), (7), (8), (9), (10), (11):

$$L(VM_i^t) = N(T,t)/S(VM_i^t). \tag{5}$$

$L(VM_i^t)$ means the load on a VM_i can be calculated as the number of tasks at the time t, $N(T,t)$ is the number of tasks at the time t in the service queue of VM_i, $S(VM_i^t)$ denotes the service rate of VM_i at the time t.

$$C(VM_i^t) = Pe_{numi} \times Pe_{mipsi} + VM_{bwi}. \tag{6}$$

Pe_{numi} is the number of processors in VM_i, Pe_{mipsi} is million instructions per second of all the processors in VM_i, VM_{bwi} is the capability of communicational bandwidth of VM_i.

$$(CC)_i = L(VM_i^t)/C(VM_i^t). \tag{7}$$

$C(VM_i^t)$ is the capacity on a VM_i at the time t.

$$L = \sum_{i=1}^{m} L(VM_i^t). \tag{8}$$

L is the loads exerted on all VMs in a DC.

$$C = \sum_{i=1}^{m} C_i. \tag{9}$$

C is the capacity of all VMs in a DC.

$$CC = L/C. \tag{10}$$

$$U = \sum_{j=1}^{n} \text{ and } C_{\max} = \max\{C_j,\ j = 1, \ldots, n\}. \tag{11}$$

U denotes maximum MS is defined as the maximum total time of completion of all workflows.

4 PROCESS SMART MPIA AND JOB MIGRATION IN THE TEXT MINING

The general process of doing work is described in such a way that after designing the general cloud model, the initial values of input parameters are valued by the job scheduler. After that, there is the waiting step for a request (flow) to enter. An all-broadcast query message is sent to all VMs that can respond to the current flow. The selection of the best VM is conducted based on the algorithms. Afterwards, the current flow is sent to the VMs using the function "AddJobVM" to execute so that its status is reported to the cloud management. The function "RetRelatedVMs" takes the ID of current flow ready on the queue and returns the VM that running the flow. In this way, the VM executes the current flow and stores the status of modes. If the query is not done, the next job will be executed. An inquiry message is sent to those flows which do the next job. Then, the selection of the best flow is carried out. At the end, the flow is sent to the selected VM. In this way, it is determined by which VM each of the different flows is executed using the algorithms. To updating the time step in this text mining work, the MPI run time of transactions in the dataset is sorted using the function "SortTransTime". The time function "AddMinutes" receives a time to generate time steps and then adds it to the current time and displays the new time in the output. The current and new

time are compared using another time function "CompareDateTime", and then the another new time is generated. In this condition, if the current time is longer than the new transaction time, then the new request is entered into the ready list. The flow executing method is that one flow is given for execution and the then next flow is received (Figure 4).

Any VM that wants to accept a flow to run and requires the cooperation with other VMs to run, it can communicate with other VMs using MPI-defined communications and send them the message whether they want to run the flow or not. As such, each cooperation VM that accepts the execution of the flow, the flow is sent to it. After running the flow by the cooperation VM, the cloud manager is informed that the flow has been completed. This is a voluntary choice of VMs to run the flows. To describe MPI smartly, each VM contains a list (or Table) of other VMs. Over time, each VM prepares a list of its neighbors based on the number of connections made to other VMs. In other words, this list contains those VMs that have more connection and send more workflows, which can help to run flows in the future. The VM that has received the flow while cannot run the flow for any reason, by checking the list of cooperators VM, it can select a cooperation VM that contains the relevant conditions to accept the flow so that it sends the flow with it. The cooperation VMs are called as AVMs in SMPIA (Figure 4).

It should be noted that if the VM did not work (e.g. the queue was full, the system crashed, the system was disconnected, etc.), in which it could not execute the current flow; as a result, the SMPIA would be applied to each of the algorithms. After updating the time step (by the time function), the list of transactions is checked. If the current time is greater than the transaction time as well as if the new transactions are entered into the list, one of them (either big or small) is chosen to be executed. On the other hand, if the small transaction is chosen, the first flow in the program is equal to 1 whereas if the big transaction is selected, the first flow is equal to 2. After transferring the flows of the transactions to the selected VMs, the SMPIA is applied and an appropriate AVM is selected based on the proposed approach. At the meantime, if the AVM is appropriate ($MS_{\min}$), the transaction is processed, and then the obtained results are saved and the stop condition is rechecked (were all transactions carried out?). Otherwise, the job migration is carried out (i.e. transferring the job from one AVM to another one) and the proposed approach is again applied to select another appropriate AVM (Figure 4).

The nfs-kernel-server and MPICH-3.0.4 packages are installed to implement MPI, due to a special feature in the master system. After that, Htop software is installed to monitor those processes running in parallel in MPI. Then, in the Hosts file, the master system is defined for IP systems. The copy of the program file was compiled in the "Mirror" folder using the MPI compiler. At the end, the program file is copied to the mirror folder and compiled it using the MPI compiler. Then, the program compiled by MPICH was executed using the following command:

```
/nfsshare$ mpirun –f hosts –n number. /MPI_sample.
```

It should be mentioned that instead of *number*, the number of processors desired to be involved with the program could be entered. Moreover, the program names are entered instead of *MPI_sample*. After that, the program run well on all systems and the performance of the processors was observed on each of the Slave computers with the help of *htop*. Ultimately, the user code was copied in the AVM buffer by MPI, and then it was prepared for parallel execution. To better demonstrate the above procedure, Figure 4 illustrates the framework of process SMPIA and job migration in text mining work. Here, Figure 5 illustrates the relationship between VMs and AVMs in VMPIB.

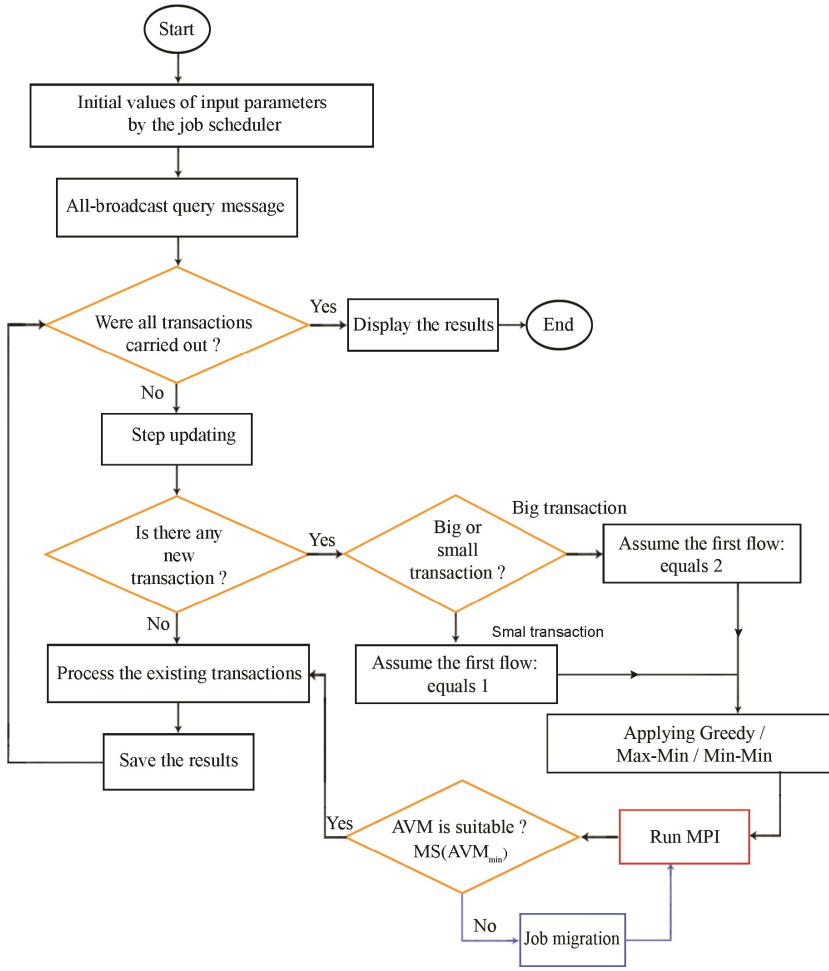

Figure 4. The framework of our process placement SMPIA and job migration

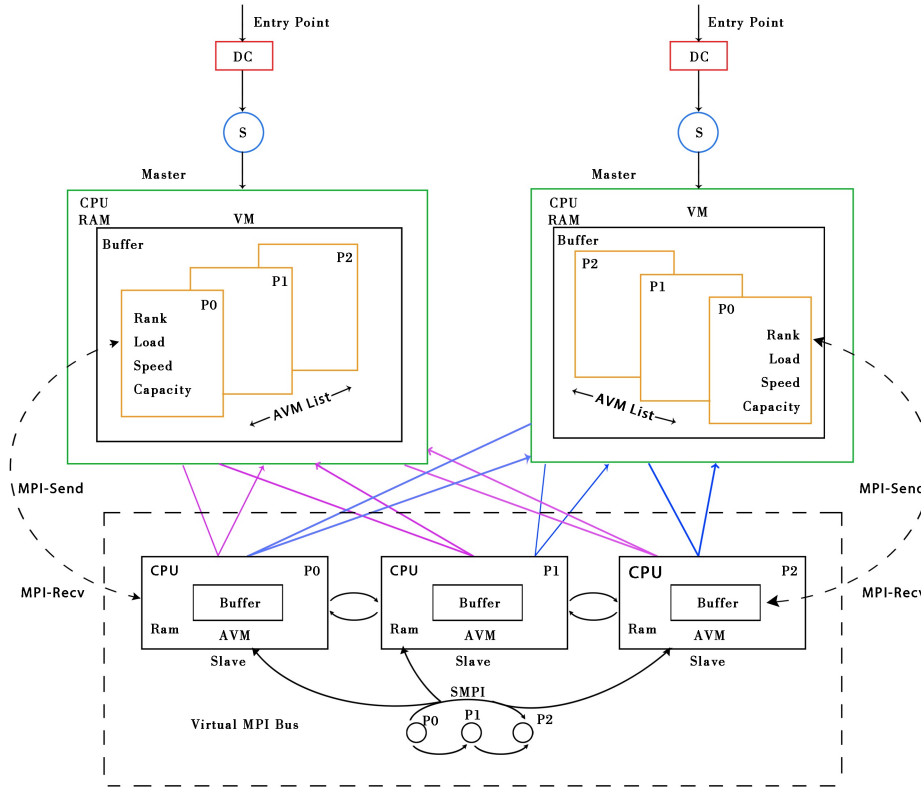

Figure 5. The relationship between VMs and AVMs in VMPIB

The SMPIA was performed in the three phases to pick up the current MPI's flows from the current VM and to transfer it to the AVM at VMPIB using the following phases: AVM rank computation phase, AVM rank-based selection phase, and MPI's flows mapping phase.

4.1 AVM Rank Computation Phase

4.1.1 Inactive MPI Process

In this case, all VMs were ranked based on the number of their connections with other VMs in the VMPIB. Note that any VM had a distributed MPI table to compute the ranking of VMs and manage MPI table. After that, the adjacent VMs were sorted out by managing the VM table based on the number of connections with other VMs in the VMPIB.

4.1.2 Active MPI Process

As discussed before, the proposed method was utilized to optimize the task scheduling and resource allocation by the MPIA (active MPI) and the non-MPIA (inactive MPI). In this way, if the number of transaction is greater than zero, a value is set for the method variable as follow: with "method = 1" the Greedy algorithm or "method = 2" the Max-Min algorithm or "method = 3" the Min-Min algorithm is determined. Note that MPI contains two processes; one is defined as Inactive MPI process due to it is executed prior the application execution; and the other is defined as Active process due to it is executed parallel user's application. Although the MPI was available in both approaches, it was smart in the proposed approach, because, with "EnableMPI = 1" the MPI is Active and "EnableMPI = 0" the MPI is Inactive. The value 1 refers to Enable MPI while the value 0 is used to Desable MPI. This paper takes advantage the Greedy, and fixed heuristic algorithms such as Max-Min, Min-Min [14, 17, 23] to find the optimal solution. These algorithms were implemented in the cloud platform on the datasets of MCITI.

In this case, all AVMs were ranked based on the number of connection with other AVM before they reach to the MPI table. The MPI table will be taken into account for VM configuration in order to choose the best AVM for current MPI's flows as well as to avoid congested of the MPI's flows. The probability of choosing an AVM for MPI's flows was determined using the Roulette Wheel selection method. According to this method, the probability of selecting AVM is equal to *the ratio of AVM rank i to total ranking of all AVMs*. The probability of selecting any $AVM_i(P_i)$ can be computed using Equation (12). Obviously, the probability of selecting an AVM with higher rank would be higher. In this matter, the MPI is defined as the communication of the VM with other AVMs in the distributed network system. In other words, the AVMs that were in touch along with most of them as well as those received more flows, were then introduced to run the next MPI's flow on the VM list. In each VM, the list of AVMs was provided in a table, based on which the minimum rank of each AVM was equal to 1. P_i or Run Times (RTs) of AVMs was calculated according to Pseudo-Code derived in Table 2.

$$P_i = \frac{\text{The ratio of AVM rank } i}{\text{Total ranking of all AVMs}} = \frac{f_i}{\sum_{j=1}^{NC} f_j}. \tag{12}$$

The Pseudo-Code of MPI algorithm is derived in Table 3.

Besides, the rank of each AVM varies based on MS's calculation. For each rank that MPI gives to the VM, $(MPI - VM_{rank})$, the rank for AVMs (AVM_{rank}) is periodically obtained according to the MS's change. As such, the pseudo-code of Send and Receive in MPI communications is provided in Table 4 for the user codes in the buffer.

```
(01)  Start
(02)  Read AVMs                          / * AVMs denotes AVMs in a data center */
        /* resource list contains AV[m] ← {AVM_1, AVM_2, AVM_3, ..., AVM_m} */
(03)  AVMsL ← Length(AVMs)
      /* AVMsL denotes the length of AVMs or the amount of current workload on
      AVMs */
(04)    For i ← 1 to AVMsL
(05)      Index ← R_AVMs(i)
                  /* Index denotes array index, R_AVMs denotes Related AVMs */
(06)      A_1 ← L_flow
                                        /* L_flow denotes MPI's flow load */
(07)      B_1 ← L_AVMs(i, Index)
                          /* L_VMs total computational loads on all of the AVMs */
(08)      C_1 ← S_AVM(Index, 4)
                                  /*S_AVM denotes size (capacity) of AVM */
(09)      D_1 ← FID
           /* FID denotes MPI's flow id, flow list contains F[f] ← {F_1, F_2, ..., F_f} */
(10)      E_1 ← ES(D_1, Index)
            /* ES denotes execution speed of the D_1 MPI's flow on AVM index */
(11)      TT ← ((A_1 + B_1)/C_1)/E_1          /* TT denotes temp time */
(12)      RT ← TT                               /* RT denotes run time */
(13)      Display "RT"
(14)    End for
(15)  End
```

Table 2. Pseudo-code for run times of AVMs

4.2 AVM Rank-Based Selection Phase

In the second phase, the list of AVMs would be checked by the MPI management and the AVM via the highest priority was selected. In this approach, the effect of selecting an AVM on the number of next flows was assessed, which would lead to an increase in their ranks. In this phase, the AVM's MS was computed as well. By comparing the obtained results of the implemented SMPIA and the applied algorithms, the effect of changing the selection of an AVM was precisely explored on both decreasing and increasing the time of MS. The best AVM was selected based on Pseudo-Code in Table 5. In Table 6, SMPIA is implemented for AVMs.

The third phase consists of SMPI functions that are responsible to allocate MPI's flows to the selected AVMs using the proposed method.

4.3 MPI's Flows Mapping Phase

In phase III, the flows were mapped onto the AVMs using these algorithms as well as the defined functions. According to the calculated MS, the flow from the VMs

(01) Start
(02) Upload MPI_{AVMs}(Table)
/* MPI_{AVMs}(Table) denotes the MPI table of AVMs */
(03) Based on the rank of each AVM, the following tasks are done, respectively.
(04) Inquire L_{AVM}
(05) Inquire ES_{AVM}
/* ES_{AVM} denotes the execution speed of current MPI's flows on the AVM */
(06) Inquire T_{CF}
/* T_{CF} denotes the execution time of current flow on the AVMs */
(07) Inquire C_{AVM} /* C_{AVM} denotes the capacity of the AVM*/
(08) Inquire C_{VM}
(09) Calculate MS_{AVM}
/* MS_{AVM} denotes the MS of the AVM for the current MPI's flow */
(10) Calculate MS_{VM}
/* MS_{VM} denotes the MS of the VM for the current MPI's flow */
(11) If $MS_{AVM} < MS_{VM}$
/* This is a scientific contribution of the paper */
(12) $AVM \leftarrow CF$ /* CF denotes current flow */
(13) $R_{AVM} \leftarrow R_{AVM} + 1$ /* R_{AVM} denotes the rank of AVM */
(14) Go to steep 19
(15) Else
(16) $R_{AVM} \leftarrow R_{AVM} - 1$
(17) Go to steep 04 /* Job migration */
(18) End if
(19) End

Table 3. Pseudo-code of MPI algorithm

via higher MS would be transferred to the AVMs via lower MS in order to reduce the mapping time (TET and completion time), resource involvement, and resource consumption. After that, the SMPIA was applied onto these algorithms. In this phase, the current flow of the current VM was removed and then mapped onto the selected AVM using the combination of the mentioned algorithms with the SMPIA. After the run of the flow accepted by the AVM, more flows were accepted by the AVM to run. In the main program, an AVM was selected to the each of input MPI's flows. Then, an ID for each of AVM was determined. Moreover, a function namely "Size" received the input MPI's flows and then calculated theirs number and capacity. The process of selecting the AVM and assigning the MPI's flow to the selected AVM was performed using a combination of these algorithms with SMPI method. Here, it should be mentioned that when the implementation of the MPI's flow was accepted by the AVM, the replication of a MPI's flow to run by the AVMs intelligently would be more. The cost function of the SMPIA calculation based on Pseudo-Code was presented as $O(P^3)$ in Table 7.

```
(01)  Start
(02)    If (SelAVM_rank < MPI − VM_rank)
(03)      {
(04)        Send (VM_cf, SelAVM_id)
(05)        Recv (VM_cf, SelAVM_id)
(06)      }
(07)    Else
(08)      {
(09)        Recv(VM_cf, SelAVM_id)
(10)        Send(VM_cf, SelAVM_id)
(11)      }
(12)    End if
(13)  End
```

Table 4. Pseudo-code of send and receive

5 SOLVING THE TSRA AND STARVATION PROBLEM USING SMPIA

First, the non-SMPIA was implemented to each of the Greedy, Max-Min and Min-Min. In the following, SMPIA were applied to each algorithm in order to assess the performance of the proposed approach in the Greedy, Max-Min, and Min-Min cloud systems. Two approaches were executed parallel to each other. Any kinds of

```
(01)  Start
(02)  Input: ORT                                   /* ORT denotes other run time */
(03)  Output: OAVM                          /* OAVM denotes other virtual machine */
(04)  ORT ← −1
      /* There is always an input to the numbers of tasks plus one, so entries: Number
         of jobs +1 */
(05)  OAVM(ID) ← −1              /* OAVM (ID) denotes the ID of Other AVM */
(06)  Read VMs
(07)  VMsL ← Length(VMs)
(08)    For i ← 1 to VMsL
(09)      If AVM_i(T_CF) < TRT
                                 /* T_CF denotes execution time of current MPI's flow */
            /* AVM_i(T_CF) denotes the execution time of current flow in AVM_i */
                                                   /* TRT denotes temp run time */
(10)        TRT ← AVM_i(T_CF)
(11)        ORT ← TRT
(12)        OAVM(ID) ← AVM_i(RT)
(13)      End if
(14)    End for
(15)  End
```

Table 5. Pseudo-code of choose the best AVMs

```
(01)  Start
(02)  Input: FID, TCL, TRT.
(03)  Output: OAVM (ID), ORT.
(04)      OAVM(ID) ← −1
      /* There is always an input to the numbers of tasks plus one, Entries: Number
      of jobs +1 */
(05)      ORT ← −1
(06)      Related_AVMsS ← Related_AVMs(FID)
      /* Related_AVMS denotes size (capacity) of AVMs, Related_AVMs denotes all
      of Related AVMs */, /* Related_AVMs denotes the related AVMs */
(07)      RT ← zeros (C (Related_AVMs))
(08)  AVMs_Count ← zeros (C (Related_AVMs))
            /* AVMs_Count denotes the number to each corresponding AVM */
(09)    For i ← 1 to AVMs_Count
(10)      Index ← Related_AVMS(i)
(11)      A1 ← TCL
                    /* TCL denotes task computation load of the MPI's flow */
(12)      B1 ← AVMs_Load(1, Index)
                          /* AVMs_Load denotes the total load of the AVM */
(13)      C1 ← AVMS(Index, 4)
                               /* AVMS denotes size (capacity) of all AVMs */
(14)      D1 ← FID
(15)      E1 ← ES(D1, Index)
(16)      TT ← ((A1 + B1)/C1)/E1
(17)      RT(i) ← TT
(18)    End for
(19)    For i ← 1 to AVMs_Count
(20)      If (RT(i) < TRT)
(21)          TRT ← RT(i)
(22)          OAVM(ID) ← Related_AVMS(i)
(23)      End if
(24)    End for
(25)  End
```

Table 6. Pseudo-code of SMPIA implementation for AVM

changes in the state of successor flows in successor VMs (changes in load, capacity, etc.) affected the next flows of subsequent AVM. The SMPIA is applied on different TSRA strategies as follows:

5.1 Solving the TSRA Problem with Applying SMPIA onto the Greedy Algorithm

In the non-MPIA, the best VM was chosen according to the lowest load variance is calculated with the Greedy algorithm for predecessor flows through the Equation (13):

```
(01)  Start
(02)  Input: IA and PRs
      /* IA denotes individual array, PRs denotes processes that are running */
(03)  Output: EV
      /* EV denotes an evaluation value function that the same as maximum MS */
(04)  TT ← EI(IA, PRs)
                                  /* EI denotes evaluation individual or temp time */
      /* There is always an input to the numbers of tasks plus one, so, the job of
      evaluation individual function is to receive an array called individual and the
      execution MPI's flows (processing requests), and calculate how much time in-
      dividual needs to execute. that's mean: Entries: Number of jobs +1, This is
      a scientific contribution of the paper */
(05)  TL ← Lengh(IA)
                                  /*  TL denotes task length of the individual array  */
                          /*  Task or transaction list contains T[t] ← {T1, T2, ..., Tt} */
                             /*  Any array is a task and any the cell of array is a job */
(06)  EV ← −1
(07)    For i ← 1 to TL
(08)      Index ← IA(i)
(09)      DO
(10)        {
(11)        A1 ← PRs(i, 4)
(12)        B1 ← LAVMs(1, Index)
(13)        C1 ← SAVM(Index, 4)
                                        /* SAVM denotes the size (capacity) of AVM */
(14)        D1 ← PRs(i, 3)
(15)        E1 ← ES(D1, Index)
(16)        TT ← ((A1 + B1)/C1)/E1
(17)        If TT > EV              /* Founding the maximum run time (MS) */
(18)          EV ← TT
(19)        End if
(20)        }
(21)      While EV! = −1
(22)        If EV ← −1
(23)          Print "it has not changed"
(24)        End if
(25)      End do
(26)    End for
(27)  End
```

Table 7. Pseudo-code of the cost function to SMPIA

$$Var(x) = \frac{\sum(x - \bar{x})^2}{m - 1}. \tag{13}$$

In the third phase for the next flows the appropriate AVM was chosen according to the calculation of the MS formula for all AVMs in the SMPIA. The shortest MS time caused the selection of one of the AVMs. By choosing AVM in VMPIB, the speed of execution tasks could be enhanced, and the mean of the MS and TET parameters were minimized for the considered workflows. The variables and definitions used in the paper are listed in Table A1 of Appendix A. IDs is assigned to AVMs by the "For Loop" of the Greedy algorithm using Pseudo-Code in Table 8. The steps of applying the SMPIA onto the Greedy algorithm in the cloud system are exhibited in Figure 6.

```
(01)  Start
(02)  Input: AVMsS, InProcessReqs
      /* AVMsS denotes the size (capacity) of the AVMs and is global variable, the
      InProcessReqs denotes the number of MPI's flows (requests) */
(03)  Output: SAVM(ID)          /* SAVM(ID) denotes ID of selected AVMs */
(04)    For ipc ← 1 to InProcessCount
                  /* InProcessCount denotes the number of the InProcessReqs */
(05)        FlowID ← InProcessReqs(ipc, 3)
(06)        AVMsCount ← Length(AVMS)
              /* AVMsCount denotes the lengths (AVMs counter) of AVMs */
(07)        For i ← 1 to AVMsCount
(08)          If (FlowID == AVMS(i, 3))
(09)            SAVM(ID) (ipc) ← i
(10)          End if
(11)        End for
(12)      End for
(13)  End
```

Table 8. Pseudo-code of assign IDs to AVMs in the Greedy-SMPIA

5.2 Solving the TSRA Problem with Applying SMPIA onto the Max-Min and Min-Min Algorithms

In the non-SMPIA, the best VM was selected with the minimum completion time onto the Max-Min and Min-Min algorithms for the predecessor flows. Meanwhile, in the third phase, the appropriate AVM was chosen for the subsequent flows according to the calculation of the MS formula for all AVMs in the SMPIA. Both the Equations (14) and (15) are calculated for the flow of k on all VMs in Max-Min and Min-Min, respectively. By calculating these equations for all VMs, the minimum value would be found, which is clear in the located VM. The variables and definitions used in the paper are listed in Table A1 of Appendix A. The steps of applying

the SMPIA onto these Max-Min and Min-Min algorithms in the cloud system are exhibited in Figure 7.

$$Fitness = (\text{Max-Min})\, VM_{\min} = \frac{L_{CF}(k) + VMLoad(i)}{ES(k,i)}, \tag{14}$$

$$Fitness = (\text{Min-Min})\, VM_{\min} = ((A_1 + B_1)/C_1)\,/E_1. \tag{15}$$

Obviously, the current flow mappings onto the chosen AVM in the Max-Min and Min-Min algorithms were almost the same, the only subtle difference was that, in the Min-Min, the flow with low execution time could be assigned to the AVM with the minimum completion time.

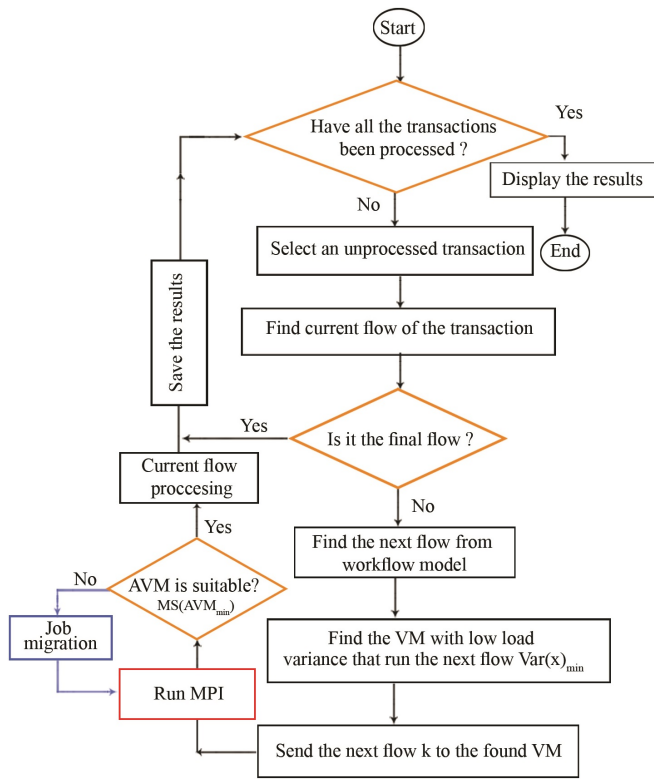

Figure 6. Flowchart of applying SMPIA to Greedy algorithm

The SMPIA was executed for Greedy, Max-Min and Min-Min algorithms based on pseudo-code in Table 9.

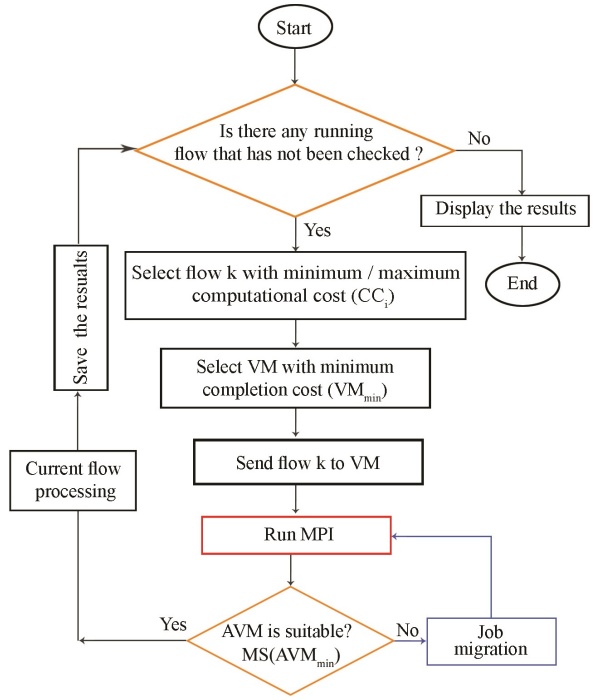

Figure 7. Flowchart of applying SMPIA to Max-Min and Min-Min algorithms

6 SOLVING THE STARVATION PROBLEM USING SMPIA

In this paper, the SMPIA is applied onto the Greedy, Max-Min and Min-Min algorithms. Then, the results of solving the starvation problem (i.e. large waiting times for small and big jobs) and the lack of sufficient resources are achieved as follows:

1. In comparison with the Greedy algorithm and the SMPIA, the best AVMs were specified in a short time so that the correspondence successor flows were executed at less time. Meanwhile, three parameters were simultaneously improved in Greedy algorithm (see Table 10).
2. In comparison with the Min-Min algorithm and the SMPIA, the TET and the completion time were considered priority. At first, those tasks were scheduled that had a minimum TET and a minimum completion time. In this way, the starvation was fixed for big tasks using the SMPIA, which was advantageous for bigger tasks in subsequent flows. The predecessor flows were processed earlier in small jobs. In addition, the subsequent flows with SMPIA were processed in big jobs earlier. Note that when the number of big tasks became more than the number of small tasks (i.e. increasing in the workload), this issue was observed

```
(01)  Start
(02)  L_SelAVMs ← (L_NF + L_SelAVMs)
                        /* L_SelAVMs denotes the load of the selected AVMs */
                          /* L_NF denotes the load of the next MPI's flow */
(03)  TRT ← (L_NF/C_SelAVM)
                        /* C_SelAVM denotes the size (capacity) of the SelAVM */
(04)  RT(SelAVM) ← TRT
                        /* RT(SelVM) denotes the run time of selected AVM */
(05)      If Enable-MPI = 1 and method = type of method (number: 1 or
          2 or 3)
                          /* Enables MPI and select the type of algorithm */
(06)          Calculate OAVM(ID), ORT          /* for next MPI's flow */
(07)          If OAVM(ID) > 0
(08)              SelAVM ← OAVM(ID)
(09)              TRT ← ORT
(10)              RT(SelAVM_i) ← TRT
                      /* RT(SelAVM_i) denotes run time of selected AVM_i */
(11)          End if
(12)      End if
(13)  End
```

Table 9. Pseudo-code of SMPIA to Greedy, Max-Min and Min-Min algorithms

noticeably. In this way, three parameters were simultaneously improved in the Min-Min (as listed in Table 10).

3. In comparison with the Max-Min algorithm and the SMPIA, the TET and completion time were prioritized as well. At first, those tasks were scheduled that had maximum TET and minimum completion time. This issue was the advantage of smaller tasks in the subsequent flows where the starvation was fixed in Max-Min for small jobs. It should be noted that the TET of the predecessor flows became longer especially for small jobs. This issue would lead to creating a change in the selection of AVMs, which was effective to decrease the minimum MS. Afterwards, the predecessor flows in small tasks were processed, but with smart MPI, then, the subsequent flows in small tasks were processed earlier. When the number of big tasks became more than the number of small tasks (i.e. increasing in the workload), this issue became noticeable. Subsequently, an increase was obtained in the number of flows of the AVMs due to the smartness of AVMs, which had minimum MS. At the meantime, the MS was reduced by changing the selection of AVMs, therefore, the performance of the Max-Min algorithm became more efficient, which was chosen to assign the next flow. Here, three parameters were improved simultaneously in the Max-Min. The greater change in AVM selection, the greater the increase or decrease in MS time would be. According to the previous discussions [38], the Minimum Completion Cloud (MCC), MEdian MAX (MEMAX) and Cloud Min–Max Normalization (CMMN)

generate a balance between MS and average cloud utilization in order to achieve a trade-off between them through solving the problem of starvation. According to discussions the SMPIA reduces the starvation problem by considering the TET and MS of the tasks. Other details are provided in Table 10.

7 EVALUATION OF SMPIA

Max-Min and Min-Min [24] and Greedy algorithms [25] were utilized extensively and successfully to map independent tasks onto resources in computational systems. They had $O(N^2 * M)$ [24] and $O(M * N)$ [25], respectively, in which N represents the number of tasks and M represents the number of processors.

In order to evaluate the proposed approach in the distributed system, some performance parameters such as TET, MS, and RU were utilized. Moreover, Greedy, Max-Min, and Min-Min algorithms were employed to investigate the proposed approach in the distributed system. The performance of the SMPIA was assessed by calculating the performance of the TET, MS, and RU parameters in both non-MPIA and SMPIA. Note that both parameters of the number of records and number of VMs were assumed to be constant. Some practical tests were implemented on the actual data in a homogenous environment including 4 DCs, 22 servers, 132 VMs, 132 flows, and 324 telecommunication equipment. In the following, programs were executed at least 50 times in a system with identical specifications to compute the average of parameters. Ultimately, the mean values were recorded as well. To do so, 201 535 records were collected on transactions (including deals and tenders) of the telephone company from 2011 to 2017. The simulation and implementation were carried out using MATLAB software. Besides, the considered experiments were done on a system the follow features: CPU 1.83 GHz, Core i7 4 GB RAM. First, the non-SMPIA was implemented to each of the Greedy, Max-Min and Min-Min. In the following, MPIA were applied to each algorithm in order to assess the performance of the proposed approach in the Greedy, Max-Min, and Min-Min cloud systems. Two approaches were executed parallel to each other. Any kinds of changes in the state of successor flows in successor VMs (changes in load, capacity, etc.) affected the next flows of subsequent AVM.

7.1 Evaluation of Total Execution Time

In this process, the execution time was calculated as the TETs using Equation (1). As illustrated in Figure 8, the TET decreases at 132 cloud workloads with SMPIA. Moreover, the maximum percent of improvement TET is 55.80 % at 132 cloud workloads in Min-Min algorithm; but SMPIA performs better than the non-SMPIA. In addition, the TET in Greedy-SMPIA and Max-Min-SMPIA improved in comparison with Greedy and Max-Min algorithms. This decrease reflects the impact of the SMPIA to optimize the TET parameter as well as to improve the performance of the proposed system. The implementation of next flows of transactions (i.e. deals and

tenders) with a minimum execution time was prioritized, due to the use of AVMs and ranking based on having the most number of connections with other AVMs. The initiating of any requests of transactions (deals and tenders) in the cloud-based system was carried out in a due time and in a short while. In this way, each request was answered in a short time. Other details are provided in Table 10.

7.2 Evaluation of Maximum Makespan

In this process, the MS was calculated as the maximum MS using Equation (2). As can be observed from Figure 9, the MS time was decreased to 132 cloud workloads with the MPIA. The maximum percent of improvement MS time is 55.94 % at 132 cloud workloads in Max-Min algorithm; but SMPIA outperforms the non-SMPIA. The maximum percent of improvement in Min-Min is 53.04 % but SMPIA performs better than non-MPIA. Furthermore, the maximum completion time of transactions (deals and tenders) in the proposed system with Greedy was less than other algorithms. The processing of each job was performed faster and completed in a short time. This decrease reflects the impact of the SMPIA to optimize the completion time parameter as well as to improve the system performance. Note that changing the choice of AVMs can be effective to reduce MS. In addition, the execution of next flows of transactions (deals and tenders) via a minimum completion time was prioritized because of the simultaneous use of AVMs and the obtained ranks based on the highest number of connections. After all, the requests for transactions (deals and tenders) were answered faster and the last jobs were completed sooner. The aforementioned results confirmed the effect of proposed approach on the appropriate distribution of load on the proposed system resources. Other details are provided in Table 10.

7.3 Evaluation of Resource Utilization

Equation (3) is formed to calculate the RU. As exhibited in Figure 10, the RU increases at 132 cloud workloads with SMPIA. Besides, the maximum percent of improvement RU is 12.28 % at 132 cloud workloads in Greedy algorithm; but SMPIA performs better than non-SMPIA in this way. The utilization of cloud resources for the Greedy has increased up to 87 % (12.28 %), which revealed the impact of the SMPIA to optimize the RU parameter. Greedy scored better value in utilization of resources than Max-Min and Min-Min. Note that any increase in the utilization of cloud resources emphases that the system was performing more efficiently using the SMPIA. Other details are listed in Table 10.

8 DISCUSSIONS AND ANALYSIS

The TET and the completion time of the workflows with the Min-Min and Max-Min algorithms were noticeably decreased using the application of MPIA. In addition,

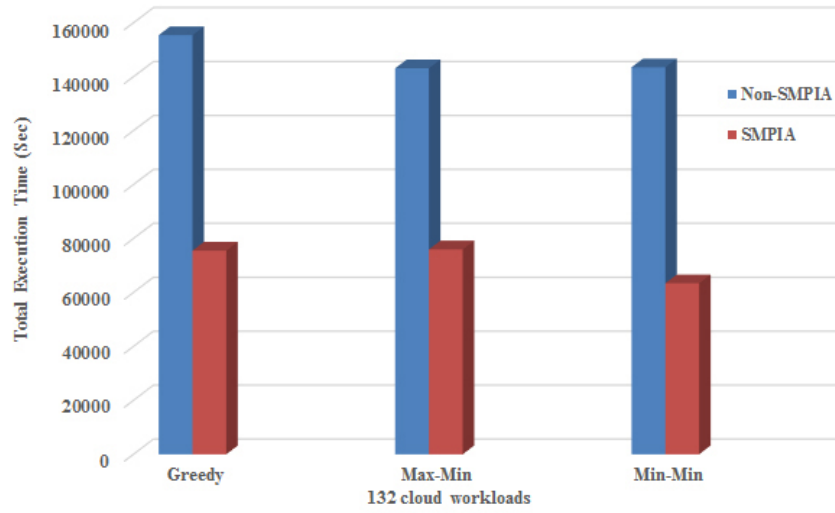

Figure 8. Effect of 132 cloud workloads on total execution time with Non-SMPIA and SMPIA

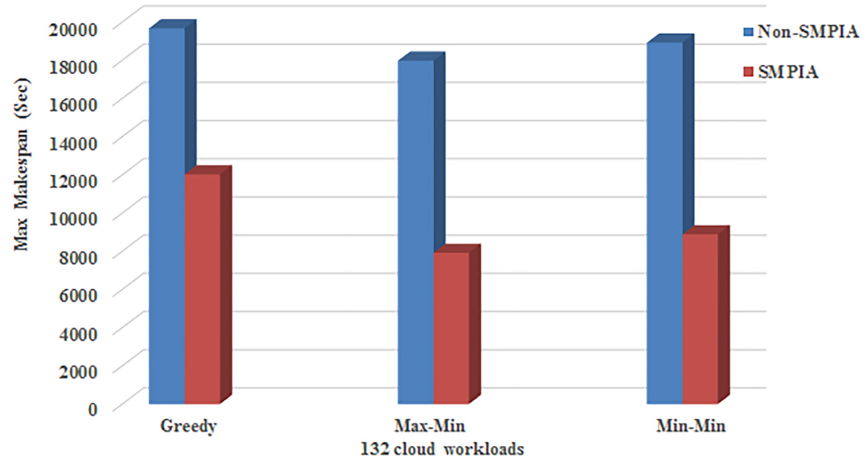

Figure 9. Effect of 132 cloud workloads on makespan with Non-SMPIA and SMPIA

Dataset	Non-SMPIA			SMPIA			Improvement (%)		
	TET	MS	RU [%]	TET	MS	RU [%]	TET [%]	MS %	RU [%]
MCITI	155 445	19 668	74.72 %	75 523	12 027	87 %	51.10 %	38.84 %	12.28 %
MCITI	143 172	17 976	75.72 %	76 008	7 919	87.52 %	49.91 %	55.94 %	11.80 %
MCITI	143 517	18 921	76.87 %	63 430	8 884	88.73 %	55.80 %	53.04 %	11.86 %

Table 10. Comparison performance parameters with Non-SMPIA and SMPIA

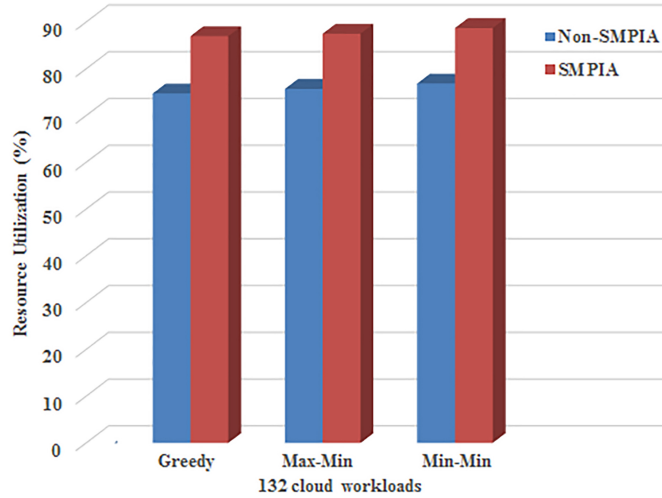

Figure 10. Effect of 132 cloud workloads on resource utilization with Non-SMPIA and SMPIA

the RU was meaningfully increased as well. It should be noted that changing the choice of an AVM to execution flows in the Min-Min and Max-Min algorithms was further evident on a number of workflows. This is due to the fact that in SMPIA, when AVM accepts the execution of the flow, it will again have a request to run by other AVMs. The effect of changing the choice of AVMs in MS time variations was increased with Min-Min and Max-Min algorithms. By choosing AVM in VMPIB, the speed of execution tasks could be enhanced, and the mean of the MS and TET parameters were minimized for the considered workflows.

The achieved results of calculating the time complexity of algorithms, the cost function of the SMPIA, and comparing the simulation, indicated that the Max-Min algorithm performance was noticeably better than the other algorithms; hence, the Max-Min algorithm was chosen to allocate the next job. Comparing the TET and MS of algorithms confirmed that the TET and MS with the SMPIA decreased, as compared to ones of the non-SMPIA. In addition, implementing the proposed approach decreased the TET from 143 517 seconds in the Min-Min algorithm to 63 430 seconds (55.80 %). Meanwhile, the MS time from the 17 976 seconds in Max-Min algorithm decreased to 7 919 seconds (55.94 %). Furthermore, the MS time in Min-Min decreased from 18 921 seconds to 8 884 seconds (53.04 %). Moreover, the RU rate in Max-Min algorithm increased from 75.72 % to 87.52 % (11.80 %). Accordingly, the executing of a workflow was purposefully enhanced. The cloud computing metrics (execution time and MS) and cloud providers (e.g. RU) were considered as part of the Multi-Objective Optimization (MOO) of real environments. Concerning the results of SMPIA in three parts, particularly those obtained with the proposed algorithm, optimal utilization of resources was provided to enhance the

system efficiency. Comparison of the results indicated the significant performance of the proposed approach by improving the efficiency and proper distribution of load in the cloud.

8.1 Resource Utilization with Total Execution Time

As illustrated in Figure 11, TET in non-SMPIA was 59.2 % greater than SMPIA at 74.72 % involvement levels and RU; nevertheless, TET in SMPIA was 59.2 % less than the non-SMPIA at 88.73 % involvement levels and RU. That is, the performance of the SMPIA was better than one of the non-SMPIA.

8.2 Resource Utilization with Makespan Time

In Figure 12, MS in non-SMPIA is 54.84 % greater than SMPIA at 74.72 % involvement levels and RU; nonetheless, MS in SMPIA is 54.84 % lesser than non-SMPIA at 88.73 % involvement levels and RU. Thus, the SMPIA outperforms the non-SMPIA in terms of the performance.

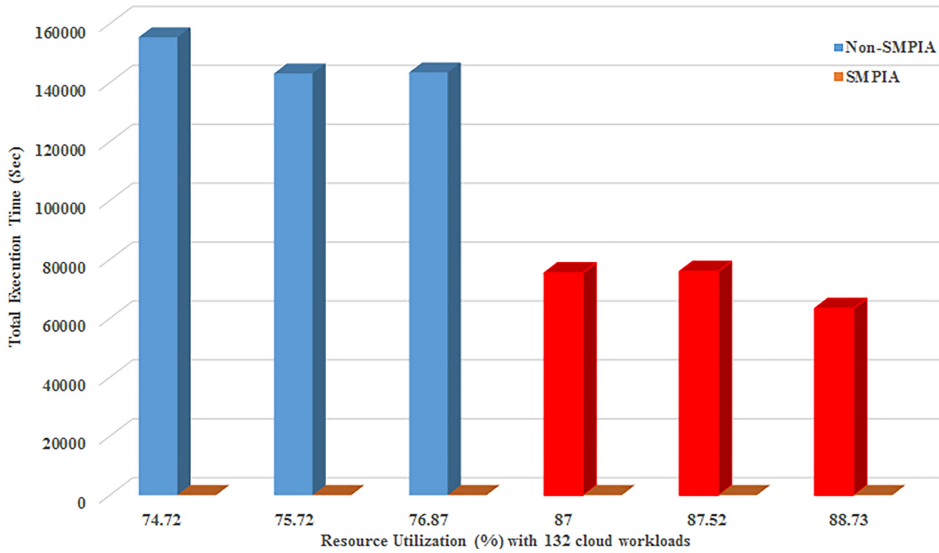

Figure 11. Effect of total execution time on resource utilization with SMPIA and non-SMPIA

9 CONCLUSIONS AND FUTURE RESEARCH

In this paper, a SMPIA with the probability of choosing the AVM was developed for the workflows using the Roulette Wheel selection method. This approach was

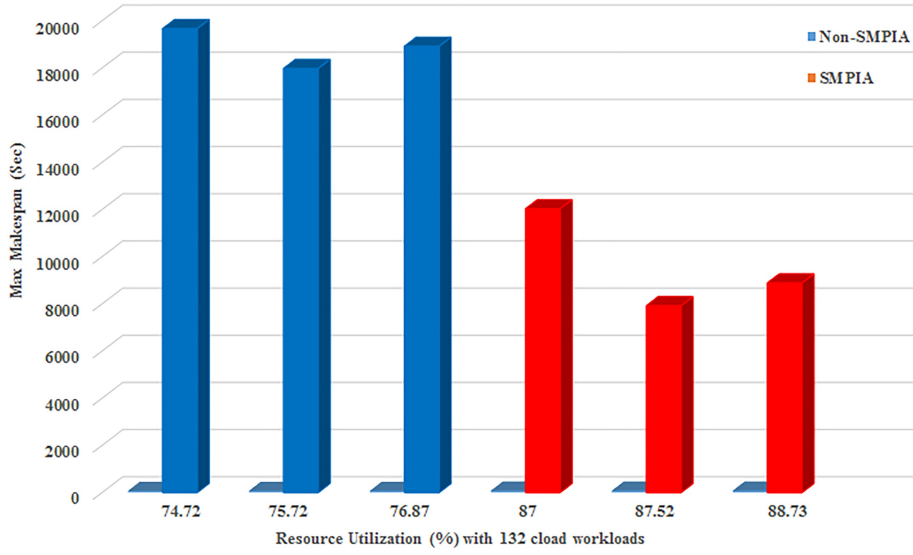

Figure 12. Effect of makespan on resource utilization with SMPIA and non-SMPIA

provided in three phases: resource-ranking, resource selection, and optimal task-resource mapping. In this way, MS and TET were significantly reduced using this approach, based on which the RU rate was simultaneously enhanced. The simulation results in the MATLAB confirmed that the minimum MS value was considered as the best solution for the optimal resource allocation and task mapping in the MPI based on the cloud resources. The best mode was belonged to Max-Min, which with SMPIA the MS up to 55.94 % and TET by up to 49.91 % improved. MS was chosen as an optimization factor, in which those solutions containing the minimum MS were chosen as the best task-resource mapping performance, which could reduce the cloud resource consumption. According to the obtained result, regarding the TET and MS of the tasks, the proposed SMPIA could reduce the starvation problem (large waiting time for small and big tasks) and the lack of sufficient resources. In addition, a multi-objective improvement approach was developed with a less complex time $O(p^3)$ for the SMPIA. An analysis of experimental actual data in the environment confirmed that the SMPIA was more efficient than the non-SMPIA and previous methods in the proposed system. As such, through the selection of AVMs and the computing the rank of them, the volume of jobs was properly distributed on the resources so that the operational efficiency of the system increased. In the following, an efficient task scheduling model was proposed using the SMPIA and Roulette Wheel selection method. However, some limitations of this paper could be the traffic congestion caused by MPI communications in the virtual network, the hidden topology of the network from the users' point of view, the delay caused through sending and receiving flows in the VMPIB. The mentioned system was

ready for the use immediately after practical application evaluation. Therefore, for future research, the performance prediction of SMPIA application will be developed using a fuzzy SMPIA to propose a minimum MS.

Acknowledgements

This project was conducted at Islamic Azad University – Sari Branch, Iran, with the cooperation of MCITI and Iran Telecommunication Company. The document of the research contract between the student, the professors, and Islamic Azad University – Sari Branch was recorded in the electronic book of Notary Public Office No. 29 of Sari under No. 15117, dated 10/07/2017.

APPENDIX

A VARIABLES AND DEFINITIONS USED IN THE PAPER

Variable	Definition
ORT	The Other Run Time is the execution time of another AVM while its initial value is equal to -1.
EI	The job of Evaluation Individual function is to receive an array called Individual and the execution flows, and calculate how much time Individual needs to execute, which is the same as the temp time.
EV	An Evaluation Value function that calculates maximum run time (MS).
ES(k, i)	The speed of execution flow$_k$ on VM_i
MS	MS is defined as completion time of the last job. Maximum MS is defined as the maximum total time of completion of all workflows. MS.
ET	Execution Time is defined as the TET of every single task from the beginning to the end.
RU	RU shows the extent of the involvement of hardware resources in the cloud to percentages, which are defined as the amount of resource involved.
TET	TET of all tasks from the beginning to the end.
TT	Temp Time, which takes to run an array called individual (the initial value is -1), $AVM_{\min}$
VMLoad(i)	The amount of current workload on AVM_i
L_{CF}	The computational Load of Current Flow.
A_1	The computational load (workloads) of the flows (process requests).
B_1	The total load of the AVMs.

C_1	The total capacity of the AVMs.
D_1	Flow type.
E_1	The execution speed of the flow on the VM.
OAVM(ID)	The Other AVM ID is the ID of another AVM and its initial value is equal to -1.
$RT(AVM_i)$	The Run Time of AVM_i.
ARU	Average RU.
$\overline{x}$	Mean values.
P	The number of execution transactions during the current time step.
m	The number of machines.
ART	Average Response Time.
ACT	Average Completion Time.
Zeros	Takes the array of the corresponding VMs and calculates matrix size (execution time) of each MPI's flow of AVMs in them and put it in the RTs.
TMLB	Task Mapping and Load Balancing.
CCS	Cloud Computing and Simulation.
NCT	Network Communication Time.
SMM	Segmented Min-Min.
WSRA	Workflow Scheduling and Resource Allocation.
ACU	Average Cloud Utilization.
MCS	Maximize Customer Satisfaction.
MCM	MPI Communication Management.
LAMPICS	Latency-Aware-MPI-Cloud-Scheduler.
MPAR	MPI-Performance-Aware-Reallocation.
IGATS	Improved Genetic Algorithm Task Scheduling.
CG	Conjugate Gradient.
NPA	Network Performance Awareness.
NPB	NAS Parallel Benchmarks.
CMPI	Cloud-MPI.

Table A1: List of variables and definitions

REFERENCES

[1] Ma, T.—Chu, Y.—Zhao, L.—Ankhbayar, O.: Resource Allocation and Scheduling in Cloud Computing: Policy and Algorithm. IETE Technical Review, Vol. 31, 2014, No. 1, pp. 4–16, doi: 10.1080/02564602.2014.890837.

[2] Gomez-Folgar, F.—Valin, R.—García-Loureiro, A. J.—Pena, T. F.—Zablah, J. I.—Ferreiro, R. V.: Cloud Computing for Teaching and Learning

MPI with Improved Network Communications. In: Mikroyannidis, A., Hernández Rizzardini, R., Schmitz, H.-C. (Eds.): Workshop on Cloud Education Environments (WCLOUD 2012). CEUR Workshop Proceedings, Vol. 945, 2012, pp. 22–27.

[3] Gomez-Folgar, F.—Indalecio, G.—Seoane, N.—Pena, T. F.—García-Loureiro, A. J.: MPI-Performance-Aware-Reallocation: Method to Optimize the Mapping of Processes Applied to a Cloud Infrastructure. Computing, Vol. 100, 2018, No. 2, pp. 211–226, doi: 10.1007/s00607-017-0573-6.

[4] Espínola, L.—Franco, D.—Luque, E.: MCM: A New MPI Communication Management for Cloud Environments. Procedia Computer Science, Vol. 108, 2017, pp. 2303–2307, doi: 10.1016/j.procs.2017.05.069.

[5] Zhuang, W.—Huang, L.: Overview of Cloud Computing Resource Allocation and Management Technology. 2019 6th International Conference on Systems and Informatics (ICSAI), Shanghai, China, 2019, pp. 713–718, doi: 10.1109/icsai48974.2019.9010101.

[6] Gong, Y.—He, B.—Zhong, J.: Network Performance Aware MPI Collective Communication Operations in the Cloud. IEEE Transactions on Parallel and Distributed Systems, Vol. 26, 2015, No. 11, pp. 3079–3089, doi: 10.1109/tpds.2013.96.

[7] He, Q.—Zhou, S.—Kobler, B.—Duffy, D.—McGlynn, T.: Case Study for Running HPC Applications in Public Clouds. Proceedings of the 19th ACM International Symposium on High Performance Distributed Computing (HPDC '10), 2010, pp. 395–401, doi: 10.1145/1851476.1851535.

[8] Amiri, M.—Mohammad-Khanli, L.: Survey on Prediction Models of Applications for Resources Provisioning in Cloud. Journal of Network and Computer Applications, Vol. 82, 2017, pp. 93–113, doi: 10.1016/j.jnca.2017.01.016.

[9] Kumar, K. D.—Umamaheswari, E.: Resource Provisioning in Cloud Computing Using Prediction Models: A Survey. International Journal of Pure and Applied Mathematics, Vol. 119, 2018, No. 9, pp. 333–342, https://acadpubl.eu/jsi/2018-119-9/articles/9/32.pdf.

[10] Rad, P.—Boppana, R. V.—Lama, P.—Berman, G.—Jamshidi, M.: Low-Latency Software Defined Network for High Performance Clouds. 2015 10th System of Systems Engineering Conference (SoSE), IEEE, 2015, pp. 486–491, doi: 10.1109/sysose.2015.7151909.

[11] Espínola, L.—Franco, D.—Luque, E.: Improving MPI Communications in Cloud. ACM-W Europe WomENcourage Celebration of Women in Computing, 2016, https://womencourage.acm.org/archive/2016/poster_abstracts/womENcourage_2016_paper_20.pdf.

[12] Antonenko, V.—Chupakhin, A.—Petrov, I.—Smeliansky, R.: Improving Resource Usage in HPC Clouds. In: Korenkov, V., Strizh, T., Nechaevskiy, A., Zaikina, T. (Eds.): Proceedings of the XXVII International Symposium on Nuclear Electronics and Computing (NEC 2019). CEUR Workshop Proceedings, Vol. 2507, 2019, pp. 180–184, http://ceur-ws.org/Vol-2507/180-184-paper-31.pdf.

[13] Xie, Z.—Shao, X.—Xin, Y.: A Scheduling Algorithm for Cloud Computing System Based on the Driver of Dynamic Essential Path. PloS One, Vol. 11, 2016, No. 8, Art. No. e0159932, doi: 10.1371/journal.pone.0159932.

[14] Almezeini, N.—Hafez, A.: An Enhanced Workflow Scheduling Algorithm in Cloud Computing. Proceedings of the 6th International Conference on Cloud Computing and Services Science (CLOSER 2016), Vol. 2, 2016, pp. 67–73, https://www.scitepress.org/Papers/2016/59083/59083.pdf, doi: 10.5220/0005908300670073.

[15] Samadi, Y.—Zbakh, M.—Tadonki, C.: E-HEFT: Enhancement Heterogeneous Earliest Finish Time Algorithm for Task Scheduling Based on Load Balancing in Cloud Computing. 2018 International Conference on High Performance Computing and Simulation (HPCS), IEEE, 2018, pp. 601–609, doi: 10.1109/hpcs.2018.00100.

[16] Gawali, M. B.—Shinde, S. K.: Task Scheduling and Resource Allocation in Cloud Computing Using a Heuristic Approach. Journal of Cloud Computing, Vol. 7, 2018, No. 1, Art. No. 4, doi: 10.1186/s13677-018-0105-8.

[17] Singhal, S.—Patel, J.: Load Balancing Scheduling Algorithm for Concurrent Workflow. Computing and Informatics, Vol. 37, 2018, No. 2, pp. 311–326, doi: 10.4149/cai_2018_2_311.

[18] Sakellariou, R.—Zhao, H.: A Hybrid Heuristic for DAG Scheduling on Heterogeneous Systems. Proceedings of IEEE 18th International Parallel and Distributed Processing Symposium, 2004, pp. 111, doi: 10.1109/IPDPS.2004.1303065.

[19] Espínola, L.—Franco, D.—Luque, E.: DA-MCM: A Dynamic Application-Aware Mechanism for MPI Communications in Cloud Environments. Proceedings of the 2018 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA), The 2018 World Congress in Computer Science, Computer Engineering and Applied Computing (CSCE'18), 2018, pp. 211–217, https://csce.ucmss.com/cr/books/2018/LFS/CSREA2018/PDP3183.pdf.

[20] Cruz, E. H. M.—Diener, M.—Pilla, L. L.—Navaux, P. O. A.: EagerMap: A Task Mapping Algorithm to Improve Communication and Load Balancing in Clusters of Multicore Systems. ACM Transactions on Parallel Computing (TOPC), Vol. 5, 2019, No. 4, pp. 1–24, doi: 10.1145/3309711.

[21] Hilman, M. H.—Rodriguez, M. A.—Buyya, R.: Task Runtime Prediction in Scientific Workflows Using an Online Incremental Learning Approach. 2018 IEEE/ACM 11th International Conference on Utility and Cloud Computing (UCC), 2018, pp. 93–102, doi: 10.1109/ucc.2018.00018.

[22] Dhinesh Babu, L. D.—Krishna, P. V.: Honey Bee Behavior Inspired Load Balancing of Tasks in Cloud Computing Environments. Applied Soft Computing, Vol. 13, 2013, No. 5, pp. 2292–2303, doi: 10.1016/j.asoc.2013.01.025.

[23] Malik, B. H.—Amir, M.—Mazhar, B.—Ali, S.—Jalil, R.—Khalid, J.: Comparison of Task Scheduling Algorithms in Cloud Environment. International Journal of Advanced Computer Science and Applications, Vol. 9, 2018, No. 5, pp. 384–390, doi: 10.14569/ijacsa.2018.090550.

[24] Tabak, E. K.—Cambazoglu, B. B.—Aykanat, C.: Improving the Performance of Independent Task Assignment Heuristics MinMin, MaxMin and Sufferage. IEEE Transactions on Parallel and Distributed Systems, Vol. 25, 2014, No. 5, pp. 1244–1256, doi: 10.1109/tpds.2013.107.

[25] Madni, S. H. H.—Latiff, M. S. A.—Abdullahi, M.—Abdulhamid, S. M.—Usman, M. J.: Performance Comparison of Heuristic Algorithms for Task Scheduling in IaaS Cloud Computing Environment. PloS ONE, Vol. 12, 2017, No. 5, Art. No. e0176321, doi: 10.1371/journal.pone.0176321.

[26] Ma, J.—Li, W.—Fu, T.—Yan, L.—Hu, G.: A Novel Dynamic Task Scheduling Algorithm Based on Improved Genetic Algorithm in Cloud Computing. In: Zeng, Q.-A. (Ed.): Wireless Communications, Networking and Applications (WCNA 2014). Springer India, New Delhi, Lecture Notes in Electrical Engineering, Vol. 348, 2016, pp. 829–835, doi: 10.1007/978-81-322-2580-5_75.

[27] Jena, T.—Mohanty, J. R.: GA-Based Customer-Conscious Resource Allocation and Task Scheduling in Multi-Cloud Computing. Arabian Journal for Science and Engineering, Vol. 43, 2018, No. 8, pp. 4115–4130, doi: 10.1007/s13369-017-2766-x.

[28] Maswood, M. M. S.—Develder, C.—Madeira, E.—Medhi, D.: Energy-Efficient Dynamic Virtual Network Traffic Engineering for North-South Traffic in Multi-Location Data Center Networks. Computer Networks, Vol. 125, 2017, pp. 90–102, doi: 10.1016/j.comnet.2017.04.042.

[29] Pande, S. K.—Panda, S. K.—Das, S.: A Customer-Oriented Task Scheduling for Heterogeneous Multi-Cloud Environment. International Journal of Cloud Applications and Computing (IJCAC), Vol. 6, 2016, No. 4, Art. No. 1, 17 pp., doi: 10.4018/ijcac.2016100101.

[30] Zhou, Z.—Zhigang, H.: Task Scheduling Algorithm Based on Greedy Strategy in Cloud Computing. The Open Cybernetics and Systemics Journal, Vol. 8, 2014, No. 1, pp. 111–114, doi: 10.2174/1874110x01408010111.

[31] Chen, H.—Wang, F.—Helian, N.—Akanmu, G.: User-Priority Guided Min-Min Scheduling Algorithm for Load Balancing in Cloud Computing. 2013 National Conference on Parallel Computing Technologies (PARCOMPTECH), IEEE, 2013, pp. 1–8, doi: 10.1109/parcomptech.2013.6621389.

[32] Liang, B.—Dong, X.—Wang, Y.—Zhang, X.: A Low-Power Task Scheduling Algorithm for Heterogeneous Cloud Computing. The Journal of Supercomputing, 2020, Vol. 76, No. 9, pp. 7290–7314, doi: 10.1007/s11227-020-03163-8.

[33] Wu, M. Y.—Shu, W.—Zhang, H.: Segmented Min-Min: A Static Mapping Algorithm for Meta-Tasks on Heterogeneous Computing Systems. Proceedings of 9th Heterogeneous Computing Workshop (HCW 2000), IEEE, 2000, pp. 375–385, doi: 10.1109/hcw.2000.843759.

[34] Hung, T. C.—Hieu, L. N.—Hy, P. T.—Phi, N. X.: MMSIA: Improved Max-Min Scheduling Algorithm for Load Balancing on Cloud Computing. Proceedings of the 3rd International Conference on Machine Learning and Soft Computing (ICMLSC 2019), 2019, pp. 60–64, doi: 10.1145/3310986.3311017.

[35] Zhang, J.—Zhai, J.—Chen, W.—Zheng, W.: Process Mapping for MPI Collective Communications. In: Sips, H., Epema, D., Lin, H. X. (Eds.): Euro-Par 2009 Parallel Processing (Euro-Par 2009). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 5704, 2009, pp. 81–92, doi: 10.1007/978-3-642-03869-3_11.

[36] Arabnejad, H.—Barbosa, J.: Fairness Resource Sharing for Dynamic Workflow Scheduling on Heterogeneous Systems. 2012 IEEE 10th International Symposium

on Parallel and Distributed Processing with Applications, 2012, pp. 633–639, doi: 10.1109/ispa.2012.94.

[37] Hsu, C. C.—Huang, K. C.—Wang, F. J.: Online Scheduling of Workflow Applications in Grid Environments. Future Generation Computer Systems, Vol. 27, 2011, No. 6, pp. 860–870, doi: 10.1016/j.future.2010.10.015.

[38] Panda, S. K.—Jana, P. K.: Efficient Task Scheduling Algorithms for Heterogeneous Multi-Cloud Environment. The Journal of Supercomputing, Vol. 71, 2015, No. 4, pp. 1505–1533, doi: 10.1007/s11227-014-1376-6.

Mehran Mokhtari is Ph.D. student in computer (software engineering, Islamic Azad University, Sari Branch, Sari, Iran). Two Master of computer (Software Engineering from Islamic Azad University, Damghan and South Tehran Branch, Damghan and Tehran, Iran). Bachelor of computer science (Mathematics: Application in Computer, Islamic Azad University, Lahijan Branch, Lahijan, Iran). He has more than 7 research projects, more than 3 books, more than 30 paper published in conference and journals. Interests: ability to programing with the network simulator-ns-2 &&3 for more than 15 years, WSN, ICT, wireless network, cloud computing, QoS in VPN, installation and implementation of OSPF protocol on network topology, project risk management, performance analysis.

Peyman Bayat is Assistant Professor and Faculty Member of Computer Engineering Department, Islamic Azad University Rasht Branch, Rasht, Iran. Ph.D. in computer (Computer Systems Engineering from University Putra Malaysia). Master of computer (Computer Software Engineering, IAU of Arak). Bachelor of electronic (Electronic Engineering, IAU of Arak). He has more than 13 research projects, more than 7 books, more than 38 articles published in journals. Interests: philosophy and arts.

Homayun Motameni is Associated Professor and Faculty Member of Computer Engineering Department. Islamic Azad University Sari Branch, Sari, Iran. Ph.D. in computer (Software from Islamic Azad University, Science and Research Branch, Tehran, Iran). Master of computer science (Machine Intelligence from Islamic Azad University, Science and Research Branch, Tehran, Iran). Bachelor of computer (Software from Shaheed Behest University of Tehran). He has more than 10 research projects, more than 6 books, more than 60 conference papers, and more than 100 articles published in journals. Interests: software engineering, performance analysis, evolutionary computation, fuzzy systems.