www.ijecs.in
International Journal Of Engineering And Computer Science ISSN: 2319-7242
Volume 6 Issue 2 Feb. 2017, Page No. 20324-20332
Index Copernicus Value (2015): 58.10, DOI: 10.18535/ijecs/v6i2.31

A Hybrid Multi Threaded Task Scheduling and Knapsack Load Balancing in Multiple Cloud Centers

C.Antony[1], C.Chandrasekar[2]
[1]Research & Development Centre, Bharathiar University,
Coimbatore – 641 046, Tamilnadu, India.
nyaantony4@gmail.com
[2]Department of Computer Science, Periyar University,
Salem–636011, Tamilnadu, India,
ccsekar@gmail.com

Abstract: ***Heuristic and task scheduling provide better scheduling solutions for cloud computing by greatly enriching in identifying candidate solutions, ensuring performance optimization and therefore reducing the make span of task scheduling. Several researchers have put forward scheduling and load balancing algorithms for cloud computing systems. However, how to reduce the response latency while efficiently utilizing detection operator mechanisms (switching between groups while scheduling with corresponding task) and reducing communication cost still remains a challenge. In this paper, a hybrid framework called, Multithreaded Locality Task Scheduling and Knapsack Load Balancing (MLTS-KLB) is constructed. The MLTS-KLB first schedules several tasks using Multithreaded Locality Parallel Task Scheduling (MLPTS) algorithm. The MLPTS algorithm gives a definition and method of achieving group synchronization. Secondly, a Knapsack Load balancing model is constructed by extending the migration based model. Then, after formulating the scheduling problems in the MLTS-KLB and bringing forward the MLPTS algorithm based Knapsack Fair Load Balancing algorithm, the efficiency of the MLTS-KLB is validated through simulation experiments. Simulation results demonstrate that the MLTS-KLB framework significantly reduce the latency time of parallel jobs and improves the average throughput of cloud computing environment by minimizing the average task waiting time compared to the state-of-the-art works.***

Keywords: Heuristic, Multithreaded, Locality Task Scheduling, Knapsack, Load Balancing, Group Synchronization

1. Introduction

In modern parallel and distributed applications, parallel jobs are handling in cloud computing environment, big data processing and so on. More complex computational jobs are said to be handled using cloud computing environment with the aid of virtual. The main problems to be addressed while handling complex computational jobs are scheduling and load balancing. Many researchers have designed solutions with this regard.

A hybrid scheme of task scheduling and load balancing was provided in [1] using on demand scheduling algorithm. This significantly resulted in the reduced performance of response time of parallel jobs. However, dynamic changes remained a major issue to be handled. To this, a dynamic workload adjustment model was designed in [2] with the aid of availability and requirement of various resources. But, parallel data processing was not said to be achieved. To achieve this, an intermediate data placement algorithm was designed in [3] that not only achieved higher overall average balancing performance but also reduced the execution time of job waiting in queue.

In order to supply user services, cloud data center hosts with large number of physical hosts. When cloud data center receives task requests, the cloud data center load gets imbalanced, compromising the timely user response. In [4], Load Balancing based on Bayes and Clustering was designed with the objective of improving the throughput and minimizing the failure number of task deployment.

Yet another method based on offload interrupt load balancing was presented in [5] to address the issues related to load balancing. However, with less efforts paid on scheduling, the communication cost remained a major issue to be addressed. With this a soft real time task scheduling algorithm was designed in [6] that not only ensured efficient scheduling but also provided means for load balancing. Meanwhile, optimization remained a breakthrough which was addressed in [7] using various factors.

Until now possible solutions to increase the performance of information system has be well researched for computation, analysis and storage. In [8], a Hyper Heuristic Scheduling Algorithm was designed to minimize the makespan of task scheduling. Budget driven scheduling algorithm for heterogeneous cloud was designed in [9] using two greedy algorithms called, Global Greedy Budget and Gradual Refinement resulting in cost effectiveness of the system. To address batch jobs in geographically distributed centers, thermal aware scheduling model was designed in [10]. A Traffic Balancing Oblivious Routing (TBOR) algorithm was designed in [11] using Cyclic Channel Dependency Graph (CDC).

In this paper, the main focus on the aforementioned problems, particularly considers a hybrid framework related to task scheduling and performing load balancing for the scheduled tasks in cloud environment. The main contribution of this paper is to present a hybrid Multithreaded Locality Task Scheduling

and Knapsack Load Balancing (MLTS-KLB) framework. With MLTS-KLB, the cloud servers schedule the jobs of all cloud users in cloud environment by designing a multithreaded locality parallel task scheduling model. With the jobs scheduled, load balancing is also said to be achieved using knapsack load balancing model.

The rest of this paper is organized as follows. Section 2 surveys related works. Section 3 introduces our task scheduling and load balancing framework. Section 4 presents experimental results. Section 5 provides a detailed discussion. Finally, Section 6 concludes the paper.

2. Related works

Virtualization technique adopted in cloud computing environment and resources deployment is performed in an efficient manner and therefore accessed through various services. With the modernization of current IT sector, the resource requirement in cloud environment is purely based on subscription. In [12], a hybrid bacterial swarm optimization algorithm was constructed to reduce the operational cost and maximize the resource utilization. Load balancing aware genetic algorithm was designed in [13] by applying Time Load Balance (TLB) model resulting in good load balancing properties between cloud users.

Load balancing and locality aware scheduling was presented in [14] based on distributed environment to improve the throughput rate. In [15], an in-depth analysis and study of load balancing techniques in cloud environment was presented. With cloud data centers comprising of several virtual machines with significantly different specifications, due to fluctuating resource usages, imbalance between servers is said to occur. To resolve this issue, an agent-based load balancing model [16] was designed for efficient load balancing in a distributed manner. A Task-based Load Balancing model using Particle Swarm Optimization was designed in [17] that resulted in significant reduction in the time taken for load balancing process compared to conventional load balancing methods.

A max-min task scheduling algorithm to realize load balancing was presented in [18] to improve the resource utilization and minimize the respond time of task. Yet another load balanced-based resource scheduling algorithm was constructed in [19]. A task scheduling algorithm based on resource scheduling was designed in [20] that not only met users' requirements but also improved the rate of throughput.

Scheduling algorithms for parallel jobs presented in [21] made effective use of two tier VMs resulted in the improvement of responsiveness that in turn significantly outperformed the most commonly used algorithms such as extensible Argonne scheduling system in a data center setting. In [22] a technique for efficient mapping of resource requests with a heuristic methodology was designed. Routing and scheduling algorithm in [23] for cloud architecture ensured minimal total energy consumption by switching off the network unused and/or information technology (IT) resources.

In [24] the author provided maximized resource utilization mechanism with optimal execution efficiency with the aid of proportional share model. In [25] various workflow scheduling algorithms were designed and compared with their counterparts in terms of characteristics and applicability for cloud scheduling. In [26] a mechanism for scheduling single tasks considering two objectives: monetary cost and completion time and dynamic scheduling of scientific workflows were proposed. A cloud scheduler [27] that considered both user requirements and infrastructure properties assuring virtual resources were hosted using physical resources that match their requirements without getting users about the details of the cloud infrastructure.

A fundamental drawback of the most existing researches is that they either considered scheduling of tasks in a distributed or dynamic manner or performed load balancing in cloud computing environment. To improve on this aspect and complete the previous works, we propose in this paper the MLTS-KLB framework which first schedules the task in an optimal manner and then performs load balancing for the scheduled task to achieve system load balancing. The proposed framework not only reduces the latency and average task waiting time but also improves the throughput rate or the task being assigned with the required resources in cloud environment.

3. Multithreaded Locality Task Scheduling and Knapsack Load Balancing

A hybrid parallel job scheduling and then balancing the load by cloud server through load balancer in cloud computing environment, called, Multithreaded Locality Task Scheduling and Knapsack Load Balancing (MLTS-KLB) is introduced. The MLTS-KLB framework begins by describing the preliminaries required and then present the detailed structure.

3.1 Preliminaries

Let us assume that the dimension of resources is 'd' and each cloud service provider's resources be expressed as a vector '$\overrightarrow{v_i} = (v_i^1, v_i^2, \dots, v_i^d)$', in which '$v_i^p$' is the '$pth$' dimensional resource that the cloud service provider 'i' has. The set of jobs that arrives at some particular time slot is further presumed to be '$i = \{1, 2, \dots, n\}$' with total jobs 'j' to be '$T_j, j = 1, 2, \dots, n$'.

If the resources consumed by job 'j_i' when executed on cloud service provider 'i' are a vector '$\overrightarrow{r_{ij}} = (r_{ij}^1, r_{ij}^2, \dots, r_{ij}^d)$', then each job is said to be only executed on one cloud service provider and cannot be further partitioned. Once a job is executed successfully on some cloud service provider, the MLTS-KLB receives the value of the job to be accomplished as successful. Here, the scheduling target in the MLTS-KLB is to maximize the total successful accomplishment of jobs with the constraint of resource capacity of each cloud service provider. Therefore, the scheduling problem for the MLTS-KLB is formulated as given below.

$$Max \sum_{j=1}^{n} T_j \sum_{i=1}^{m} CU_{ij} \quad (1)$$

$$subject\ to \sum_{j=1}^{n} \overrightarrow{v_{ij}}\ CU_{ij}, where\ CU_{ij} \in [0,1] \quad (2)$$

From (1) and (2), the MLTS-KLB framework is formulated as a multidimensional '$0, 1$' knapsack problem, where '1' assumes that the cloud service provider is assigned with a specific task whereas '0' assumes that the cloud service provider is not assumed with a specific task. Besides, in order to make the MLTS-KLB work longer, all the jobs arriving is said to be executed in a uniform manner on the cloud servers.

This makes the cloud users consume their resources evenly to avoid the phenomenon that certain cloud users with heavy load consume their resources too early and have to leave the system. In order to solve this problem, a scheduling algorithm based on the Multithreaded Locality Parallel Task is designed in the MLTS-KLB framework to enhance the performance of task scheduling. Meanwhile, the load of 'pth' dimensional resource for cloud service provider 'i' is then given as below.

$$Load_{in} = \frac{\sum_{p=1}^{n} r_{ij}^p CU_{ij}}{v_{ij}^p} \quad (3)$$

The following part of this section presents MLPTS algorithm, which contains the group synchronization and optimal scheduling achieved through multithreaded parallel scheduling (MPS). With the scheduled task, load balancing is said to be achieved through knapsack load balancing model.

.

3.2 Multithreaded Locality Parallel Task Scheduling (MLPTS)

In MLPTS, threads are allocated to the cloud server that utilizes multithreading architecture to perform all the scheduling processes. Each thread is assigned to a group of jobs, executed by multiple cloud centers. The objective of using MLPTS algorithm in the MLTS-KLB is the group synchronization. Such groups' synchronization in the MLTS-KLB is achieved through using multithreading parallel scheduling. It uses multithreaded parallel scheduling on a group of cloud users that sees to that when more than one thread try to access a shared resource, measures are taken such that the resource is used by only one thread at a time and therefore providing group synchronization.

This method of proceeding with the process of scheduling makes it more suitable and practical because the multithreading parallel scheduling tries to identify the best cloud user to be assigned with a corresponding job. Figure 1 shows the diagrammatic representation of multithreaded parallel scheduling (MPS).

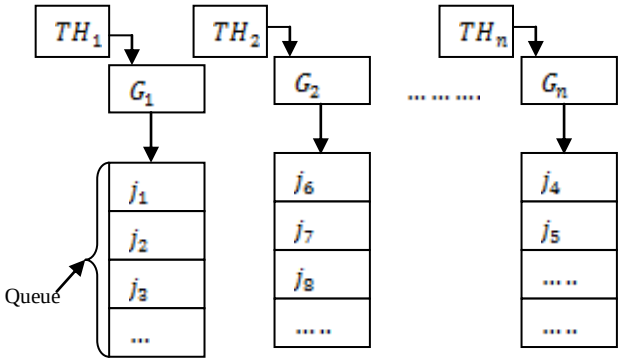

Figure 1: Multithreaded Parallel Scheduling

Let us consider multithreaded structured with threads '$Th_i = Th_1, Th_2, \dots, Th_n$' assigned to different groups '$G_i = G_1, G_2, \dots, G_n$' whose jobs to be allocated are in the corresponding group. From the figure, the jobs waiting in queue in group 1 is 'G_1', whereas the jobs waiting in queue in group 2 is 'G_2' and jobs waiting in queue in group n is 'G_n' respectively.

As shown in the figure, the MPS starts by dividing the entire set of cloud servers '$CS_i = CS_1, CS_2, \dots, CS_n$' into '$n$' groups. Each group comprises of a number of jobs to be assigned by the cloud servers. Each group is assigned with a thread that in turn schedules the jobs in the queue.

When any job is to be processed, the threads start searching in their groups. Each thread takes information about current job and starts searching in their groups. Upon successful identification of a thread that finds the resource to be assigned for the specific task, it immediately declares other threads to prevent searching for this task and starts searching for next tasks. In this way, all the cloud user's corresponding tasks are assigned with the resources by the cloud servers in an optimized manner.

Input: Resource dimension 'd', cloud service provider resource vector $\overrightarrow{v_i} = (v_i^1, v_i^2, \dots, v_i^d)$ ', job 'j_i', Cloud Users '$CU_i = cu_1, cu_2, \dots, cu_n$', Cloud Servers '$CS_i = cs_1, cs_2, \dots, cs_n$', Thread '$Th_i = Th_1, Th_2, \dots, Th_n$',

Group '$G_i = g_1, g_2, \dots, g_n$'
Output: Optimal scheduling of jobs and improved throughput
1: Begin
2: For each resource dimension 'd'
3: For each Cloud Servers 'CS_i' and Cloud Users 'CU_i' with 'i' jobs to be scheduled
4: Divide entire set of cloud servers '$CS_i = CS_1, CS_2, \dots, CS_n$' into '$n$' groups
5: Repeat
6: For each Thread 'Th_i'
7: For each groups 'G_i'
8: Allocate jobs
9: If 'current job' = 'requested job'
10: Perform searching in corresponding group
11: Upon matching prevent searching in other groups
12: End if
13: Else
14: If 'current job' <> 'requested job'
15: Perform searching in other group
16: End if
17: End for
18: End for
19: Until (all jobs are assigned to the cloud users)
20: End for
21: End for
22: End

Algorithm 1 MLPTS algorithm

The MLPTS algorithm is illustrated in algorithm 1, where the initial parameters include '$d, \underset{v_i}{\rightarrow}, j_i, CU_i, CS_i, Th_i, G_i$' respectively. Step 2 to step 4 comprises of conditional settings used that ranges from number of cloud users waiting, number of cloud servers assigned with and dividing the cloud servers in to groups. Step 8 and step 9 performs the task of assigning thread to each group and jobs scheduled to each group, with jobs scheduled in the form of queue. Conditional matching is performed from step 10 to step 16 where the current job and requested jobs are tested with and scheduled accordingly. In this way, an optimal scheduling of job is ensured by performing a perfect match.

3.3 Knapsack Load Balancing Model

Once optimal scheduling is accomplished using MLPTS algorithm, efficient load balancing has to be performed for the scheduled task. Efficient load balancing not only results in the increase of resource utilization for existing cloud users but also accommodates more cloud users in cloud computing environment. The MLTS-KLB uses a combinatorial optimization model with the aid of knapsack called as the knapsack load balancing model.

The combinatorial optimization model includes both system persistence and optimized load balancing. One of the important issues to be solved while operating load balanced service is system persistence, where the response towards user's request (i.e. job response) is stored in a specific cloud server and avoiding subsequent requests in different cloud servers. In this case, the system persistence is solved in the MLTS-KLB using knapsack handler. The MLTS-KLB framework sees to that the jobs kept across multiple requests for the same cloud user in a user's session does not search or jump into another cloud server and therefore addressing system persistence. With the system persistence mode, knapsack load balancing model is designed in the next section. Figure 2 shows the structure of knapsack load balancing model.

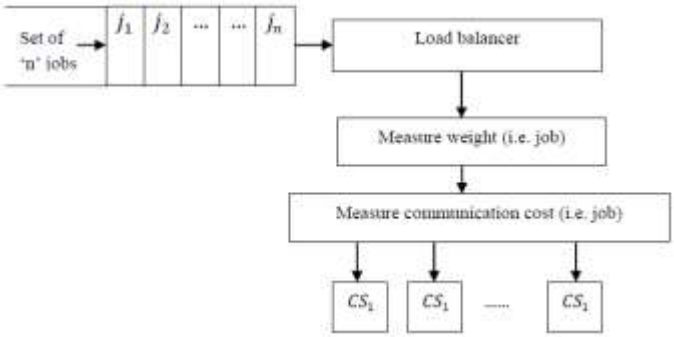

Figure 2: Knapsack load balancing

As shown in the figure, in knapsack problem there is a knapsack with cloud servers 'CS_k' and set 'N' of 'n' jobs to fill the knapsack. Each job 'i' has a prescribed weight '$Weight_i$' and communication cost '$CCost_i$' associated with it and is denoted mathematically as given below.

$$Weight_i CCost_i \geq 0 \quad (4)$$

From (4), it is assumed that for a given set of jobs, the total weight is less than or equal to a given limit of the knapsack and the communication cost has to be minimized. In the other words, the objective is to identify a subset of jobs where total size or weight is bounded by 'CS_k' and whose communication cost is minimized.

Let's further assume that there are 'n' cloud users, where '$N = \{1, 2, \dots n\}$', and each user has a '$weight = w_1, w_2, \dots, w_n$' (i.e. size) and a '$value = v_1, v_2, \dots, v_n$' (i.e. profit). The weight or size of the cloud user represents the resources the cloud user needs and is formulated as given below.

$$Weight\ (CU_i\ (t)) = R_{CU_i}(t) = R_{CPU}\ (t) + R_{MEM}\ (t) \quad (5)$$

From (5), '$R_{CPU}\ (t)$' represents the CPU requirements of cloud user 'CU_i' and '$R_{MEM}\ (t)$' represents the memory requirement

of cloud user at time 't'. On the other hand, the value or communication cost is obtained as given below.

$$CCost_i(t) = \sum_{i=1}^{n} CCost\ (CU_i CS_i) \quad (6)$$

From (6), the communication cost '$CCost_i$' at time 't' is measured on the basis of communication between the cloud user 'CU_i' and cloud server 'CS_i' respectively. With the weight and communication cost obtained the MLTS-KLB designs an algorithm to find the optimal solutions while performing load balancing.

Many researchers have used hybrid model to find the optimal solutions and the MLTS-KLB is influenced by them including the works in [1] [2] [3]. However, compared with existing researches, the Knapsack Fair Load Balancing algorithm not only considers the weight involved (i.e. CPU and memory requirements), but also the communication cost incurred, decreasing the latency time compared to the state-of-the-art works. The algorithm is illustrated in algorithm 2.

Input: Cloud Users '$CU_i = cu_1, cu_2, \ldots, cu_n$', Cloud Servers '$CS_i = cs_1, cs_2, \ldots, cs_n$', Threshold '$R_{TH}$', Time '$t$'
Output: Fair load balancing and reducing latency time
1: Begin 2: For each Cloud Servers 'CS_i' and Cloud Users 'CU_i' with 'j_i' jobs to be scheduled 3: Measure weight for each job using (5) 4: If $Weight\ (CU_i\ (t)) < TH$ 5: Measure communication using (6) 6: Update cloud server not overloaded 7: Update job assigned with required resource achieving load balancing 8: End if 9: If $Weight\ (CU_i\ (t)) > TH$ 10: Update cloud server overloaded 11: Update job not assigned and Load balancing not performed 12: End if 13: End for 14: End

Algorithm 2: Knapsack Fair Load Balancing Algorithm

In the MLTS-KLB, an algorithm named Knapsack Fair Load Balancing (KFLB) is proposed to minimize the communication cost and therefore the latency while performing load balancing for the scheduled jobs in cloud computing. KFLB is shown in algorithm 2. It consists of three parts, measuring scheduled job arrivals, evaluating jobs weight and evaluating the communication cost between the cloud user and cloud server for each job to be allocated. The scheduled job arrivals are first measured based on a queue structure. Followed by this, each jobs weight and communication cost between the cloud users in cloud computing environment is measured. Based on the resultant values, a fair load balancing is said to be achieved.

4. Experimental Settings

In this section, the experimental setup for designing Multithreaded Locality Task Scheduling and Knapsack Load Balancing (MLTS-KLB) framework in cloud computing that is used in our experiments uses the JAVA platform with CloudSim simulator is explained. The experiments were conducted on Amazon's EC2 infrastructure using the Amazon Access Samples dataset and Landsat 8 data on AWS to experiment the different parameters that offer distinct resource configurations for virtual machine instances.

The attribute details included in Amazon Access Samples dataset comprises of four categories of attributes namely, PERSON_{ATTRIBUTE}, [RESOURCE_{ID}], [GROUP_{ID}], [SYSTEM_SUPPORT_{ID}] in table 1. On the other hand, the Landsat 8 data on AWS [28] consists of a raster file containing global information for bands 1 through 11 for Landsat 8 Operational Land Imager (OLI) and Thermal Infrared Sensor (TIRS) in table 2.

Table 1: Amazon Access Samples table

Attributes	*Description*
PERSON_{ATTRIBUTE}	This category describes the 'user' who was given access. The [PERSON_ID] column is the primary key column for the file. There is one row per user.
RESOURCE_{ID}	This category of attributes represents the resources that a users can possibly have access to. A user will have a 1 in this column if the have access to it otherwise it will be 0.
GROUP_{ID}	This category of attributes represents the groups that a users can possibly have access to. A user will have a 1 in this column if the have access to it otherwise it will be 0.
SYSTEM_SUPPORT_{ID}	This category of attributes represents the system that a user can possibly be supporting. A user will have a 1 in this column if the have can possibly be supporting it, otherwise it will be 0.

Table 2: Landsat imagery used for remote estimation of lake clarity

Path	*Rows*	*Acquisition date*	*Satellite sensor*
12	27 – 30	8/30/2010	Landsat 5 TM
12	27 – 30	9/14/2004	Landsat 5 TM
12	27 – 30	9/6/1995	Landsat 5 TM
11	28 – 29	8/9/2005	Landsat 5 TM
11	27 – 29	8/9/2002	Landsat 7 ETM+
11	27 – 29	9/5/2009	Landsat 5 TM

The upcoming cloud computing environments and several application services makes the cloud users to focus on precise cloud system design with different processor types and varying ranges. Amazon EC2's interface minimizes the time required for various instances according to the changes observed in

computing requirements. MLTS-KLB experiments with the c1.medium, a compute optimized instance type, a 32-bit processor, 1.7 GB RAM and 350 local disk storage.

The objective of CloudSim is to provide a global and extensible simulation framework that compares the proposed Multithreaded Locality Task Scheduling and Knapsack Load Balancing (MLTS-KLB) framework in cloud computing with the existing hybrid task scheduling and load balancing scheme called DeMS [1] and novel Adaptive Task Scheduling strategy based on Dynamic Workload Adjustment (ATSDWA) [2] in cloud environment. Experiment is conducted to measure and evaluate the MLTS-KLB framework on the factors such as latency time, throughput rate and average task waiting time with respect to different number of jobs being assigned in cloud environment.

The latency time or mean response time for assignment of parallel jobs is measured on the basis of the mean scheduling time and the scheduling time for 'n' jobs in the queue. The mean scheduling time is the overall time required to respond to the 'n' jobs in cloud environment. Meanwhile, time for scheduling is measured on the basis of each job, where time differs for each job due to the request response rate variance for each resource requirement.

So, job assigned and the time for scheduling the assigned jobs is considered for obtaining the latency time. Let us consider a simulation scenario with 50 jobs to be assigned in the cloud environment with the mean scheduling time 'MST' being 0.58ms. Then, the latency time is measured as given below.

$$LT = \sum_{i=1}^{n} MST - (j_i * Time\ (Scheduling)) \quad (6)$$

Throughput is a measure of how many jobs are processed in a given amount of time in cloud environment. Throughput has been a measure of the comparative effectiveness in cloud environment that run many cloud user programs or accesses several cloud user jobs concurrently. Therefore, the throughput rate in the MLTS-KLB is the number of cloud users allocated with the jobs by the cloud servers in a given time period. The rate of throughput is measured in terms of percentage (%).

The average task waiting time is the product of job requests made by the cloud user to the cloud server in cloud environment and the time taken by the cloud server to respond to each jobs in the queue. The mathematical evaluation of the average task waiting time is formulated as given below.

$$ATWT = \sum_{i=1}^{n} j_i * Time\ for\ each\ job \quad (7)$$

From (7), '$ATWT$',measures the average task waiting time with respect to the jobs 'j_i' in the queue. The average task waiting time is measured in terms of milliseconds (ms).

5. Result Analysis

To better understand the effectiveness of the proposed Multithreaded Locality Task Scheduling and Knapsack Load Balancing (MLTS-KLB) framework, extensive experimental results are reported in figure 3. The MLTS-KLB framework is compared against the existing hybrid task scheduling and load balancing scheme called DeMS [1] and novel Adaptive Task Scheduling strategy based on Dynamic Workload Adjustment (ATSDWA) in cloud environment. CloudSim simulator is used to measure and experiment the factors by analyzing the percentage of result with the help of graph values. Results are presented for different number of jobs assigned. The results reported here confirm that with the increase in the number of jobs assigned, the latency time also gets increased.

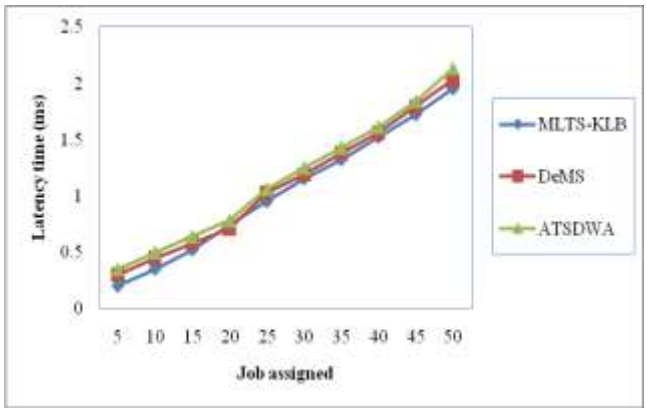

Figure 3: Comparison of latency time

Figure 3 presents the variation of latency time with respect to job assigned. All the results provided in figure confirm that the proposed MLTS-KLB framework significantly outperforms the other two methods, DeMS [1] and ATSDWA [2]. The latency time is reduced in the MLTS-KLB framework that schedules the job using multithreaded architecture. At the same time, the curve is found to be linear, which ensures that all the jobs are assigned with equal priority and therefore, an increase in the job being assigned results in the increased latency time. With the application of multithreaded parallel scheduling, group synchronization is resolved where more than one thread waits for the other to release the thread and waiting indefinitely. In MLTS-KLB framework, the jobs to be accessed in stored in the form of group, with the availability of the job in a group, the corresponding job is scheduled to the corresponding cloud user. On the other hand, the job is searched in the same block and unavailability of the job makes a search to be performed by the cloud server in other group. Therefore, the latency time for scheduled is reduced using the MLTS-KLB framework by 11% compared to DeMS and 20% compared to ATSDWA.

The targeting results of through rate using MLTS-KLB framework with the aid of Access Samples dataset and Landsat 8 dataset is provided in figure 4 and compared with two state-of-the-art methods [1], [2] based on the job assigned for scheduling in cloud environment.

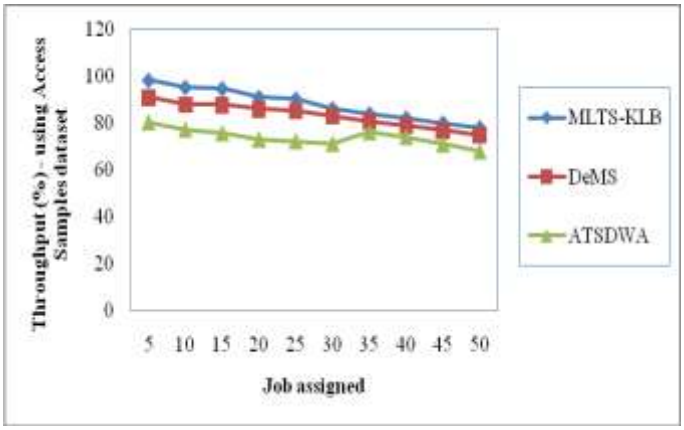

Figure 4: Comparison of Throughput with Respect To Job Assigned Using Access Samples Dataset

The throughput rate using MLTS-KLB framework with two state-of-the-art methods [1], [2] in figure 4 and figure 5 is presented for visual comparison based on the jobs assigned by cloud users in cloud environment using two different datasets. The targeting results of throughput using Access Samples and Landsat 8 dataset with respect to 50 jobs is provided in figure 4 and figure 5 respectively.

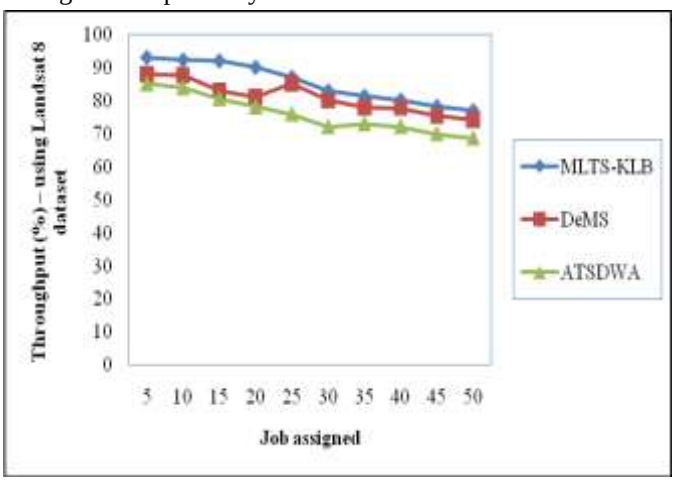

Figure 5: Comparison of Throughput with Respect to Job Assigned Using Landsat 8 Dataset

Using both the datasets, the MLTS-KLB framework ensures higher throughput rate than the existing state-of-the-art works. The MLTS-KLB framework differs from the DeMS [1] and ATSDWA [2] in that we have incorporated the MLPTS algorithm in MLTS-KLB framework that minimizes the cloud server waiting time and in turn improves the rate of throughput rate by 5 % (using Access Samples dataset) compared to DeMS. In addition, with the cloud service provider's resources being stored in a vector form of data structure, allocated in contiguous memory and also have the advantage of growing in nature. Therefore, resources scheduled and released by the cloud user is stored explicitly and the availability of resources ensures higher throughput rate using MLTS-KLB framework by 5% compared to DeMS and 16% compared to ATSDWA (using Access Samples dataset). As a result, optimum job scheduling is ensured close to the target output (i.e. average success ratio). This in turn improves the average throughput rate using MLTS-KLB framework by 5% (using Landsat 8) compared to DeMS and 11% (using Landsat 8) compared to ATSDWA respectively.

To explore the influence of average task waiting time on MLTS-KLB framework with the help of Knapsack Fair Load Balancing (KFLB) algorithm, the experiments were performed by varying the jobs assigned in figure 6.

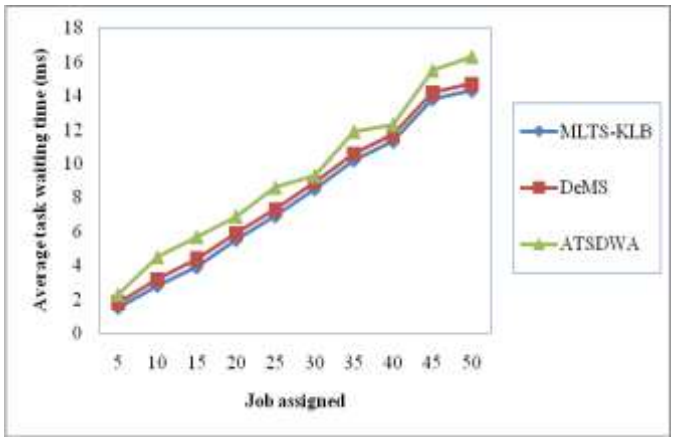

Figure 6: Comparison of Average Task Waiting Time

It also shows that with the application of KFLB algorithm it extensively provides competitive results compared to the state-of-the-art methods, namely DeMS [1] and ATSDWA [2]. The average task waiting time using MLTS-KLB framework increases with the increase in the jobs being assigned. But, comparatively performs better than the state-of-the-art methods. This is because the knapsack load balancing model using MLTS-KLB framework uses a combinatorial optimization model. This combinatorial optimization model helps in addressing system persistence through which the search for resources being made by the cloud user through jobs are restricted if found in a group. In this way, system persistence is achieved using MLTS-KLB framework and therefore reduces the average task waiting time by 8% compared to DeMS. Furthermore, optimal solutions for load balancing are achieved by comparing the value of the cloud user weight with that of the threshold. This in turn reduces the average task waiting time through balanced load by 27% compared to ATSDWA respectively.

6. Conclusion

In this paper, Multithreaded Locality Task Scheduling and Knapsack Load Balancing (MLTS-KLB) is provided based on the novel hybrid model combining Multithreaded Locality Parallel Task Scheduling and Knapsack Load Balancing model in cloud computing. This framework reduced average task waiting time and improves the throughput rate of jobs being process in cloud environment. As the framework uses the multithreaded parallel scheduling algorithm, it reduces the average task waiting time through addressing group synchronization avoiding the job being locked and resource being used by only one thread at a time. As a result, the MLTS-KLB framework performs efficient job scheduling in a parallel

manner. By applying the knapsack load balancing model in MLTS-KLB framework, load balancing is achieved using a combinatorial optimization model in cloud environment. Finally, with the application of two algorithms, MLTS and Knapsack Fair Load Balancing, latency time is reduced. A series of simulation results are performed to test the latency time, average task waiting time and throughput rate based on the jobs being assigned. Experiments conducted on varied simulation runs shows improvement over the state-of-the-art methods. The results show that MLTS-KLB framework offers better performance with an improvement of throughput rate by 8% and reduces the average task waiting time by 17% compared to DeMS and ATSDWA respectively.

References

[1] Yu Liu,Changjie Zhang, BoLi, Jianwei Niu, "DeMS: hybrid scheme of task scheduling and load balancing in computing clusters", Elsevier, Journal of Network and Computer Applications, May 2015, Pages 1-8.

[2] Xiaolong Xu, Lingling Cao, and Xinheng Wang, "Adaptive Task Scheduling Strategy Based on Dynamic Workload Adjustment for Heterogeneous Hadoop Clusters", IEEE Systems Journal, Volume 10, Issue 2, June 2014, Pages 471 – 482.

[3] Zhuo Tang, Xiangshen Zhang, Kenli Li, Keqin Li, "An intermediate data placement algorithm for load balancing in Spark computing environment", Elsevier, Future Generation Computer Systems, July 2016, Pages 1-15.

[4] Jia Zhao, Kun Yang, Xiaohui Wei,Yan Ding, Liang Hu, Gaochao Xu, "A Heuristic Clustering-based Task Deployment Approach for Load Balancing Using Bayes Theorem in Cloud Environment", IEEE Transactions on Parallel and Distributed Systems, Volume 27, Issue 2, February 2015, Pages 305 – 316.

[5] Luwei Cheng and Francis C.M. Lau, "Offloading Interrupt Load Balancing from SMP Virtual Machines to the Hypervisor", IEEE Transactions on Parallel and Distributed Systems, Volume 27, Issue 11, March 2016, Pages 3298 – 3310.

[6] Huangning Chen and Wenzhong Guo, "Real-Time Task Scheduling Algorithm for Cloud Computing Based on Particle Swarm Optimization", Springer, Cloud Computing and Big Data, January 2016, Pages 141-152.

[7] Nidhi Bansal, Amit Awasthi and Shruti Bansal, "Task Scheduling Algorithms with Multiple Factor in Cloud Computing Environment", Springer, Information Systems Design and Intelligent Applications, February 2016, Pages 619-627.

[8] Chun-Wei Tsai, Wei-Cheng Huang, Meng-Hsiu Chiang, Ming-Chao Chiang, and Chu-Sing Yang, "A Hyper-Heuristic Scheduling Algorithm for Cloud", IEEE Transactions On Cloud Computing, Volume 2, Issue 2, April-June 2014, Pages 236-250.

[9] Yang Wang and Wei Shi, "Budget-Driven Scheduling Algorithms for Batches of MapReduce Jobs in Heterogeneous Clouds", IEEE Transactions On Cloud Computing, Volume 2, Issue 3, September 2014, Pages 306-319.

[10] Marco Polverini, Antonio Cianfrani, Shaolei Ren, and Athanasios V. Vasilakos, "Thermal-Aware Scheduling of Batch Jobs in Geographically Distributed Data Centers", IEEE Transactions On Cloud Computing, Volume 2, Issue 1, March 2014, Pages 71-84.

[11] Pengju Ren, Michel A. Kinsy and Nanning Zheng, "Fault-Aware Load-Balancing Routing for 2D-Mesh and Torus On-Chip Network Topologies", IEEE Transactions on Computers, Volume 65, Issue 3, June 2015, Pages 873 – 887.

[12] V. Jeyakrishnan, P. Sengottuvelan, "A Hybrid Strategy for Resource Allocation and Load Balancing in Virtualized Data Centers Using BSO Algorithms", Springer, Wireless Personal Communications, July 2016, Pages 1-23.

[13] Zhi-Hui Zhan, Ge-Yi Zhang, Ying-Lin, Yue-Jiao Gong, and Jun Zhang, "Load Balance Aware Genetic Algorithm for Task Scheduling in Cloud Computing", Springer, Simulated Evolution and Learning, 2014, Pages 644-655.

[14] Ke Wang, Kan Qiao, Iman Sadooghi, Xiaobing Zhou, Tonglin Li, Michael Lang, Ioan Raicu, "Load-balanced and locality-aware scheduling for dataintensive workloads at extreme scales", Wiley & Sons, Journal Concurrency and Computation: Practice & Experience archive, Volume 28, Issue 1, January 2016, Pages 70-94.

[15] Geethu Gopinath PP, Shriram K Vasudevan, "An in-depth analysis and study of Load balancing techniques in the cloud computing environment", Elsevier, Procedia Computer Science, Volume 50, 2015, Pages 427-432.

[16] J. Octavio Gutierrez-Garcia, Adrian Ramirez-Nafarrate, "Agent-based load balancing in Cloud data centers", Springer, Cluster Computing, September 2015, Volume 18, Issue 3, Pages 1041–1062.

[17] Fahimeh Ramezani, Jie Lu, Farookh Khadeer Hussain, "Task-Based System Load Balancing in Cloud Computing Using Particle Swarm Optimization", Springer, International Journal of Parallel Programming, October 2014, Volume 42, Issue 5, Pages 739–754.

[18] Yingchi Mao, Xi Chen and Xiaofang Li, "Max–Min Task Scheduling Algorithm for Load Balance in Cloud Computing", Springer, Intelligent Systems and Computing, Volume 255, 2014, Pages 457-465.

[19] Haihua Chang and Xinhuai Tang, "A Load-Balance Based Resource-Scheduling Algorithm under Cloud Computing Environment", Springer, New Horizons in Web-Based Learning, 2011, Pages 85-90.

[20] Yiqiu Fang, Fei Wang, and Junwei Ge, "A Task Scheduling Algorithm Based on Load Balancing in Cloud Computing", Springer, Verlag Berlin Heidelberg, 2010, Pages 271-277.

[21] X. Liu, C. Wang, B.B. Zhou, J. Chen, T. Yang, and A.Y. Zomaya, "Priority-based consolidation of parallel

workloads in the cloud", IEEE Transactions on Parallel and Distributed Systems, Vol. 24, September 2013.

[22] C. Papagianni, A. Leivadeas, S. Papavassiliou, V. Maglaris, C.C Pastor, and A. Monje, "On the optimal allocation of virtual resources in cloud computing networks", IEEE Transactions on Computers, Vol. 62, June 2013.

[23] J. Buysse, K. Georgakilas, A. Tzanakaki, M.D. Leenheer, B. Dhoedt, and C. Develder, "Energy-efficient resource-provisioning algorithms for optical clouds", Journal of Optical Communications and Networking, Vol. 5, March 2013.

[24] Sheng Di, and Cho-Li Wang, "Dynamic optimization of multiattribute resource allocation in self-organizing clouds", IEEE Transactions on Parallel and Distributed Systems, Vol. 24, March 2013.

[25] L. F. Bittencourt, E.R.M. Madeira, and N.L.S. Fonseca, "Scheduling in hybrid clouds" Cloud Computing: Networking and Communications Challenges, IEEE Communications Magazine, September 2012.

[26][26] H.M. Fard, R. Prodan, and T. Fahringer, "A truthful dynamic workflow scheduling mechanism for commercial multicloud environments", IEEE Transactions On Parallel And Distributed Systems, Vol. 24, June 2013.

[27]Imad M. Abbadi and Anbang Ruan, "Towards trustworthy resource scheduling in clouds", IEEE Transactions on Information Forensics and Security, Vol. 8, June 2013

[28] Ian M. McCullough, Cynthia S. Loftin, Steven A. Sader," Combining lake and watershed characteristics with Landsat TM data for remote estimation of regional lake clarity", Remote Sensing of Environment, Elsevier, Mar 2012

Author Profile

Antony C received M.E degree in Computer Science and Engineering from Anna University Chennai, India in 2007. Now he is pursuing his Ph.D degree in the Department of Computer Science Bharathiar University. His research interests include Cloud Computing, Computer Security.

Chandrasekar C received his Ph.D degree in Computer Science from Periyar University Salem in 2005. He is currently Professor in the Periyar University, India. His research interests include Real-Time Computing, Cloud Computing, Advanced Computer Architecture.