RSDC (RELIABLE SCHEDULING DISTRIBUTED IN CLOUD COMPUTING)

Arash Ghorbannia Delavar,Mahdi Javanmard , Mehrdad Barzegar Shabestari and Marjan Khosravi Talebi

Department of Computer, Payame Noor Universtiy, PO BOX 19395-3697, Tehran, IRAN
a_ghorbannia@pnu.ac.ir , info@javanmard.com ,
shabestari.sanjesh@gmail.com , khosravitalebi5@gmail.com

ABSTRACT

In this paper we will present a reliable scheduling algorithm in cloud computing environment. In this algorithm we create a new algorithm by means of a new technique and with classification and considering request and acknowledge time of jobs in a qualification function. By evaluating the previous algorithms, we understand that the scheduling jobs have been performed by parameters that are associated with a failure rate. Therefore in the proposed algorithm, in addition to previous parameters, some other important parameters are used so we can gain the jobs with different scheduling based on these parameters. This work is associated with a mechanism. The major job is divided to sub jobs. In order to balance the jobs we should calculate the request and acknowledge time separately. Then we create the scheduling of each job by calculating the request and acknowledge time in the form of a shared job. Finally efficiency of the system is increased. So the real time of this algorithm will be improved in comparison with the other algorithms. Finally by the mechanism presented, the total time of processing in cloud computing is improved in comparison with the other algorithms.

KEYWORDS

Scheduling Algorithm, Cloud Computing, RSDC, PPDD

1. INTRODUCTION

In 2000 years, conditions improved in quality as the essential component systems have been established, but to shape the quality of infrastructure must provide hardware and software and the hardware and software infrastructure can be associated with conditions. The problem is that we can increase the concurrency and transaction system for high quality we have [2]. 2000 scientific experts from the raise edge technology in mechanized systems seek to change with new methods and algorithms will be able to provide technology in organizations with information technology education levels increase [3]. Information technology is an attempt to integrate combined methods for providing a suitable solution that increases customer satisfaction level is [2]. Environmental conditions for entering customers are among powerful managerial tools which are used in information technology [10]. The main idea behind cloud computing is not a new one. John McCarthy envisioned that computing facilities will be provided to the general public like a utility. NIST definition of cloud computing: Cloud computing is a model for enabling convenient, on-demand network access to a shared pool of configurable computing resources (e.g. ,networks, servers, storage, applications, and services) that can be rapidly provisioned and released with minimal management effort or service provider interaction [4]. In different cases, there are many

DOI : 10.5121/ijcsea.2012.2301

examples of using cloud computing for simulation. The first application is a Bones Structure Simulation (Bones) which is used to simulate the stiffness of human bones. The second application is a Next Generation Sequencing Analysis Workflow (GSA) for mRNA. This workflow is based on the whole shotgun sequencing [5].

One of algorithms that be used in cloud computing is genetic algorithm. Indeed , genetic algorithms are resolution systems base of computer and they use calculation models in some basic elements of gradual evolution in design and performance [9]. In the grid and cloud environments is a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capabilities [6]. Scheduling is an important tool for production and engineering. In distributed systems such as grids, that there are various sources in them, deciding about task regularity and selecting the computing node is done by a part of system calls scheduler. The purpose of this part is maximizing the efficiency and minimizing the costs [7]. Scheduling a group of independent tasks on heterogeneous processed resources in grid and cloud has been studied by a lot of researchers. And to obtain the optimal answer in the acceptable time by the heuristic search techniques for scheduling grid and cloud resources, such as GA, Tabu search, simulated Annealing [8]. Parallel and distributed heterogeneous computing has become an efficient solution methodology for various real world applications in science, engineering and business. One of the key issues is how to partition and schedule jobs that arrive at processing nodes among the available system resources so that the best performance is achieved with respect to the finish time of all input tasks [1]. Here the optimality, involving solution time and speed up, is derived in the context of specific scheduling policy and interconnection topology. The formulation usually generates optimal solution via a set of linear recursive equations. The model takes into account the heterogeneity of processor and link speeds as well as relative computation and communication intensity. Front end processors allow a processor to both communicate and compute simultaneously by assuming communication duties [1].

So due to defects in systems that have been studied, the proposed algorithm (RSDC) can help us improve scheduling problem. Now we are going to study the works that is done in scheduling.

2. PPDD ALGORITHM

Parallel and distributed heterogeneous computing has become an efficient solution methodology for various real world applications in science, engineering, and business.

One of the key issues is how to partition and schedule jobs/loads that arrive at processing nodes among the available system resources so that the best performance is achieved with respect to the finish time of all input tasks.

To efficiently utilize the computing resources, researchers have contributed a large amount of load/task scheduling and balancing strategies in the literature. Recent efforts have been focused on resource sharing and coordination across multi-site resources (multiple supercomputer centers or virtual organizations).

For divisible load scheduling problems, research since 1988 has established that the optimal workload allocation

and scheduling to processors and links can be solved through the use of a very tractable linear model formulation, referred to as Divisible Load Theory (DLT). DLT features easy computation, a schematic language, equivalent network element modeling, results for infinite sized networks and numerous applications.[1]

This theoretical formulation opens up attractive modeling possibilities for systems incorporating communication and computation issues, as in parallel, distributed, and Grid environments.

Here, the optimality, involving solution time and speedup, is derived in the context of a specific scheduling policy and interconnection topology.

The formulation usually generates optimal solutions via a set of linear recursive equations. In simpler models, recursive algebra also produces optimal solutions.

The model takes into account the heterogeneity of processor and link speeds as well as relative computation and communication intensity.

DLT can model a wide variety of approaches with respect to load distribution (sequential or concurrent), communications (store and forward and virtual cut-through switching) hardware availability (presence or absence of front end processors).

Front end processors allow a processor to both communicate and compute simultaneously by assuming communication duties.

A recent survey of DLT research can be found in . The DLT paradigm has been proven to be remarkably flexible in handling a wide range of applications.

Since the early days of DLT research, the research has spanned from addressing general optimal scheduling problems on different network topologies to various scenarios with practical constraints, such as time-varying channels, minimizing cost factors, resource management in Grid environments and distributed image processing. Load partitioning of intensive computations of large matrix-vector products in a multicast bus network was theoretically investigated in. Research efforts after 1996 particularly started focusing on including practical issues such as, scheduling multiple divisible loads, scheduling divisible loads with arbitrary processor release times in linear networks, consideration of communication startup time, buffer constraints.[1]

Some of the proposed algorithms were tested using experiments on real-life application problems such as image processing , matrix-vector product computations, and database operations. Various experimental works have been done using the divisible load paradigm such as for matrix-vector computation on PC clusters and for other applications on a network of workstations.

Recent work in DLT also attempted to use adaptive techniques when computation needs to be performed under unknown speeds of the nodes and the links. This study used bus networks as the underlying topology. Beaumont et al. consolidates the results for single-level tree and bus topologies and presents extensive discussions on some open problems in this domain.

A few new applications and solutions in DLT have been investigated in recent years. Bioinformatics, multimedia streaming, sensor networks, economic and game-theoretic approaches.

Although most of the contributions in DLT literature consider only a single load originated at one processor, scheduling multiple loads has been considered in.

Work presented in considers processing divisible loads originating from an arbitrary site on a arbitrary graph.

However, there is considered merely a single-site multi-load scheduling problem and don't address multiple loads originated at arbitrary multiple sites/nodes in networks.
The point of load origination imposes a significant influence on the performance.

In addition, when one considers multiple loads originating from several nodes/sites, it becomes much more challenging to design efficient scheduling strategies.

This study investigated load scheduling and migration problems without synchronization delays in a bus network by assuming that all processors have front-ends and the communication channel can be dynamically partitioned. Front-ends are communication coprocessors, handling communication without involving processors so that communication and computation can be fully overlapped and concurrent.

In this case, load distribution without any consideration of synchronization delay is quite straightforward as will be shown later.[1] However, in practice, it would be unreasonable to assume that the channel can be dynamically partitioned.

Especially, in the application of distributed sensor systems, front-end modules may be absent from the processing elements.

Recently, PPDD algorithm investigates the case of two load origination sources in a linear daisy chain architecture. The divisible load scheduling problems with multiple load sources in Grid environments have been studied in.

PPDD algorithm discusses about a general load scheduling and balancing problem with multiple loads originating from multiple processors in a single-level tree network.

This scenario happens commonly in realistic situations, such as applications in distributed real-time systems, collaborative grid systems (where each virtual organization can be abstracted as a resource site or a local hierarchical network), and in general load balancing and sharing applications.

In Grid environments, the proposed model can be applied to the following scenario: PPDD algorithm has a super-scheduler across multiple sites and local schedulers for each site; multiple jobs are submitted to local schedulers and possibly partitioned and migrated across multiple sites by the super scheduler for resource sharing, load balancing, and high performance throughput.
PPDD algorithm addresses a realistic situation in a distributed network of processors where in the computational load can originate at any processor on the network.

Thus, when there is more than one load to be processed in the system, unless a clever strategy for load distribution is carried out, the processors may not be efficiently utilized. In the existing DLT literature, processing multiple loads on distributed networks is addressed; however, it was assumed that all the loads originate at the central scheduler (bus controller unit in the case of bus networks). The formulation considers loads originating at different processors on the network.

PPDD algorithm proposed load distribution and communication strategies for the case when the processors are equipped with front-ends. For the case with front-ends, the algorithm simply uses to obtain the loads exchanged among processors to obtain the final load distribution l_i , i = 1, 2, . . ., K.

The PPDD algorithm takes advantage of the optimality principle to minimize the overall processing time. The PPDD algorithm guarantees to determine the near-optimal solution in finite number of steps.[1]

Since the load partitioning phase does not account the communication delays that will be encountered during the actual load communication, the algorithm obtains the near-optimal solution.
When there is considered the actual load transferring to the processors, PPDD algorithm is guaranteed to produce a near-optimal solution for homogeneous systems.

However, in heterogeneous systems use only the communication speed of the corresponding receiving link, thus causing imprecise results (when the actual load transfer takes place).

This minor discrepancy in the results is due to the fact that communication speeds are different between the active sending and receiving links, whereas the actual load transferring time is determined solely by the slowest link and not by the receiving link.

One may relaxes this assumption and considers the combined effect of both the link delays in PPDD algorithm, however, the ultimate solution may not be drastically different from what is proposed by PPDD under the current model. A significant advantage of PPDD in its current form is its simplicity to design and implement a scheduler at the root processor p0.

However, one may attempt to use other strategies to schedule and transfer loads among processors to minimize the processing time.

In these works, scheduling strategies for multiple loads arriving at a bus controller unit (BCU) were studied. However, in PPDD algorithm, there is considered the case where multiple loads originate at different sites. In any case, the algorithm can apply the previous scheduling strategies in the literature to the problem context.

At first, there is considered a single-level tree network with only one load originating on a processor. Using the following equations, the algorithm obtains a near-optimal load distribution for a single load.

Then, PPDD algorithm may repeat this procedure for all the loads residing at other sites. For the case with front-ends one may follow a similar procedure.

Thus, to balance the load among all processors in such a way that it is finished processing at the same time, is the fraction of load L assigned to processor p_i.

Another algorithm is RSA algorithm which is an extension of the load scheduling strategy for a single load. Note that when the algorithm schedules a load among all processors, processors which are not engaged in communication can process its load independently. In every iteration, in a step-by-step fashion, each processor distributes its load among all processors.

Further, each processor attempts to balance all the loads among all the processors such that they finish processing at the same time.

To compare the time performance of RSA and PPDD strategies, it is presented an example through numerical analysis. On comparison of the solutions using these two load scheduling strategies, it is observed that the overall processing time using RSA is much greater than that obtained using PPDD algorithm.

It is also observed that, in RSA strategy, the last processor to distribute is P_1 and it is the first one to finish processing its load.

The reason is that, at the last iteration, P_1 lets all other processors finish processing its remaining load at the same time while other processors have their own load during that time. At the end of the load communication phase, the remaining load at P_1 is smallest than before. Thus, all the other processors need more time to finish their loads than P_1 after the end of load communication.
A natural improvement is to repeat round-robin scheduling until the finish times of all processors are sufficiently close.[1] But RSA cannot avoid any additional time delays (overhead) incurred due to shuttling of load from and to the same processor. a load fraction transferred from P_i to P_j in previous iterations may be transferred back to P_i from P_j or P_k , thus wasting the communication resources.

Since there is no front-end to overlap communication and computation, such kind of unnecessary load “wandering” greatly prolongs the overall processing time.

On the other hand, RSA always needs m iterations to obtain the final solution while PPDD algorithm needs only (m − k) iterations. For example RSA needs five iterations while PPDD algorithm needs only one iteration to obtain a better solution.

If they improve RSA through repeating the round-robin scheduling, RSA needs more iterations to obtain a better solution. However, even in this case, improved version of RSA cannot avoid load wandering from and back to a processor either.

In PPDD algorithm it have been addressed the problem of scheduling strategies for divisible loads originating from multiple sites in single-level tree networks.

The formulation presented a general scenario with multi-site divisible loads, demanding several processors to share their loads for processing.

It is designed a load distribution strategy and communication strategy to carry out the processing of all the loads submitted at various sites. A two phase approach is taken to attack the problem. a load partitioning phase and the actual communication of load fractions to the respective processors (communication strategy). In the first phase, it is derived the near-optimal load distribution; in the second phase, it is considered the actual communication delay in transferring the load fractions to the processors, by assuming that the overall delay is contributed by the slowest link between the sending and receiving processors.

As a first step, one can relax this assumption and analyze the performance and the proposed scheduling strategies are flexible in adapting to such relaxed assumptions. For the case with front-ends, they propose a scheduling strategy, PPDD algorithm, to achieve a near-optimal processing time of all loads.

Several significant properties of PPDD algorithm are proven in lemmas and detailed analysis of time performance of PPDD algorithm was conducted.

The above analysis is also extended to homogeneous systems wherein they have shown that the time performance of PPDD algorithm with respect to various communication-computation ratios.

To implement the load distribution strategy obtained through PPDD algorithm, it is proposed a simple load communication strategy. It was demonstrated that the overall processing time obtained using PPDD algorithm is sufficiently close to the result following the actual load communication strategy proposed.[1]

To further demonstrate the efficiency of PPDD algorithm, it is also compared the time performance of PPDD algorithm with another algorithm, Round-robin Scheduling Algorithm (RSA).

The proposed load scheduling strategies can be readily extended to other network topologies in a similar way. Another interesting extension is to further study the case with multiple load arrivals at each processor, which models dynamic scheduling scenarios in grid or cloud computing environments.[1] PPDD (Processor – Set Partitioning & Data Distribution Algorithm) With Front End algorithm is presented as follows.

We consider a single-level tree network with a root processor P_0, also referred to as a scheduler for the system, and m processors denoted as $P_1, \dots, P_m$ connected via links $l_1, \dots, l_m$, respectively [1]. P_0 is considered as a router. If we do not schedule each of the loads among the set of processors, then the overall processing time of all the loads is determined by the time when the last processor finishes processing its own load.

In order to minimize the overall finish time, we should carefully re-schedule and balance the loads among all processors. Of course the processing time of each processor is clear. PPDD algorithm is described in Table 1 [1].

Table 1. PPDD algorithm [1]

Initial stage: From (1) and (2), we obtain the initial delimiter K which separates the sender and receiver sets. Load distribution: The load assigned to p_i is $l_i = L_i + \Delta L_i$, $i = 1, 2, \dots, K$ and $l_i = L_i - \Delta L_i$, $i = K+1, K+2, \dots, m$. Overall processing time: The finish time for processor p_i is given by, $T_i(m) = l_i E_i$. Thus, we obtain the overall processing time $T(m) = \max\{T_i(m)\}$, $i = 1, 2, \dots, m$.

In PPDD algorithm that mentioned in Table 1 the final processing rate for each processor is determined. This algorithm is much more efficient than other algorithms such as RSA algorithm. But the initial processing time that is considered for each processor is not the actual processing time. Because the important factors such as the time to process and the time to prepare for the process is not considered. In fact, the time that has been used as the initial time for load processing is considered without attention to the request and acknowledge time. What we did in this article is adding the request and acknowledges time to the load processing time, which is an important factor in calculating the run time.

3. RSDC ALGORITHM

RSDC algorithm is presented in cloud computing environment as follows. In our formulation, we consider a single-level tree network with m processors and a scheduler P_0. Each processor P_i has its own divisible load of size H_i to process. The goal is to design an efficient scheduling strategy to minimize the overall processing time of all the loads by partitioning and distributing the loads among all the m processors [1].

T_i (m), the total processing time at P_i, is a function of TP_i, the processing time of each processor, and l_i, the link time over the processors, and H_i, the amount of load for each processor, that is considered H_iTP_i for the processing time of each processor and H_il_i for the link time over processors. In network, generally, the final time depends on the lowest link. So if $l_i \leq l_j$ then we assume that the link time taken to reach the destination via the links li and l_j is simply H l_j. However it may be noted that is assumption does not affect the way in which the strategy is designed. In fact we will show that this assumption eases analytical tractability without loss of generality.

Since a divisible load is assumed to be computationally intensive, a natural assumption is that the computation time for a given load is much larger than the link time that is $TP_i > l_i$. In this strategy we consider that the link time between processor loads are minimized. So in RSDC algorithm we consider only processing time, not link time, for calculating the final time of process. Also we consider the request and acknowledge time, too. The strategy involves two phases. In the first phase the entire set of loads are partitioned and distributed. In the second phase the partitioned loads are transferred from one processor to another following a communication strategy. These two phases will be carried out for with front end cases. In the first phase, the scheduler P_0 collects all load distribution information about all slave processors and applies the algorithm to obtain the optimal load partitions. In the second phase the set of overload processors initiate sending data and the set of under load processors initiate receiving data. The scheduler coordinates these slaves sending or receiving operations by routing data among them. Note that although the algorithm appears iterative to obtain the optimal data partition, the amount of load migration or partition for each processor will be adjusted only once.

We consider m processors: p_1, p_2, p_3,…, p_m. p_0 processor works as a router. If the loads are not scheduled, the final processing time is equal to the time that the last processor finishes its loads. To minimize the final time, we must schedule and balance job allocation between processors. Scheduling strategy is to allocating more jobs to rapid processors and fewer jobs to slow processors. Until the volume of jobs and the processing time on different processors are not identified, we face a complex issue to find the optimal solution. In job distribution, the important point is transferring additional loads from overload processors to under load processors.

Of course the processing time for all the processors is determined. This mechanism is used to divide the loads. How to divide and use the loads is presented in Figure 1.

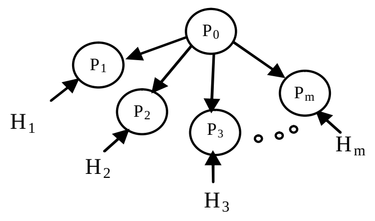

Figure 1. Single-level tree attribution of loads

In the previous proposed algorithm including PPDD algorithm, just the running time of processors has been considered whilst in the algorithm that is proposed in this paper (RSDC) request and acknowledge time is considered, too. RSDC flowchart is presented in Figure 2.

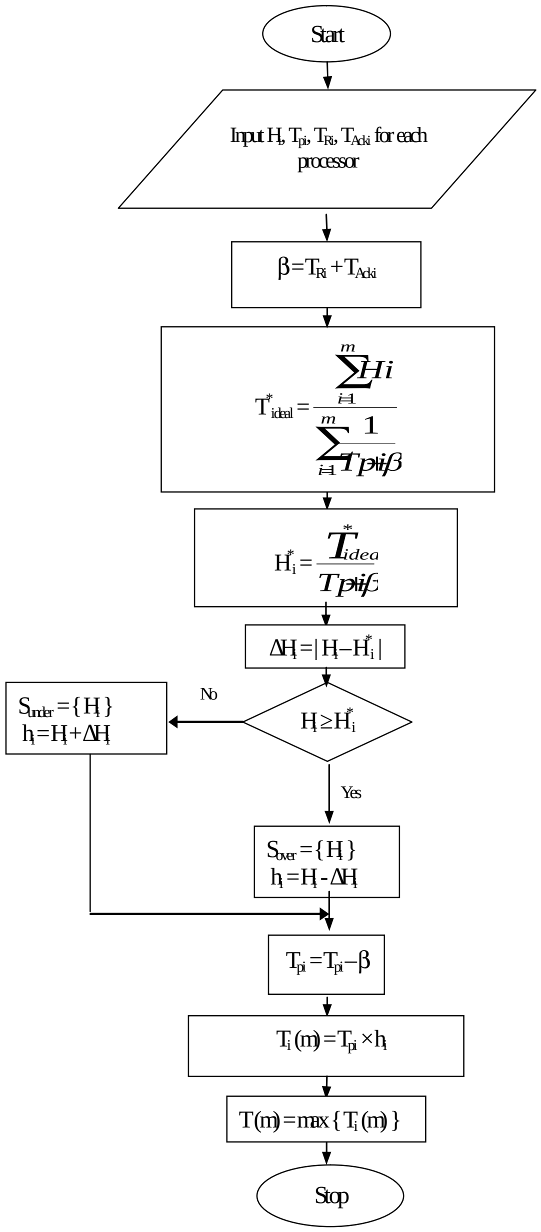
Start
Input H_i, T_{pi}, T_{Ri}, T_{Acki} for each processor
$\beta = T_{Ri} + T_{Acki}$
$T^*_{ideal} = \frac{\sum_{i=1}^{m} H_i}{\sum_{i=1}^{m} \frac{1}{T_{pi}+\beta}}$
$H^*_i = \frac{T^*_{ideal}}{T_{pi}+\beta}$
$\Delta H_i = |H_i - H^*_i|$
$H_i \geq H^*_i$
No
$S_{under} = \{H_i\}$
$h_i = H_i + \Delta H_i$
Yes
$S_{over} = \{H_i\}$
$h_i = H_i - \Delta H_i$
$T_{pi} = T_{pi} - \beta$
$T_i(m) = T_{pi} \times h_i$
$T(m) = \max\{T_i(m)\}$
Stop

Figure 2. RSDC Flowchart

Following items are used in flowchart.

m	: number of processors
H_i	: initial loads for each processor
TP_i	: processing time for each processor
TR_i	: request time for each processor
T_{Acki}	: acknowledge time for each processor
H^*_i	: average processing for each processor
ΔHi	: $H_i - H^*_i$ the deviation from the average
S_{under}	: the set of under load processors
S_{over}	: the set of overload processors
h_i	: the amount of balanced loads for each processor
T_i (m)	: the balanced processing time for each processor
T (m)	: the final time of processing the processors

First the loads and the processing time for each processor should be considered. Then with the care of processing time and request and acknowledge time, the average amount of loads should be calculated ($T^*_{ideal}(m)$). Afterwards the average of loads and the deviation from the average for each processor are calculated. Finally if the initial processing rate for each processor was higher than the average it means that there is overload and it is placed in S_{over}. And if the initial processing rate was lower than the average it is placed in S_{under}. Now, according to the algorithm, the new loads for processors of S_{over} and S_{under} are calculated. And finally the time for each processor is computed by using the following formulas.

$$\beta = T_{Ri} + T_{Acki}$$

$$T_{Pi} = T_{Pi} - \beta \qquad (1)$$

$$T_i\,(m) = T_{Pi} \times h_i$$

Considering the fact that the used time in PPDD algorithm has some problems, we could improve it by the proposed algorithm and using indicator parameters in qualification function. The purpose of the present algorithm is minimizing the time of starvation gap and maximizing the use of the system. During the work, it is possible that a processor has no data for processing. In this situation the processor may hold idle without any special activity. This event brings starvation gap for other processors because the rest of the processors, which have data for processing, want to start the processing faster. This algorithm relives the processors with fewer loads and the processors without loads from unemployment with dividing loads among processors and also relives the processors with extra loads from hard working.

4. Implementation of RSDC algorithm

After implementation of the algorithm in a programming environment and using data tables below, the following results were obtained. Table 2 determines data set for RSDC and also PPDD algorithms. Measurement unit of loads is MB and measurement unit of processing time and request and acknowledge time is millisecond.

Table 2. First data set using in RSDC and PPDD algorithms

H_i	T_{Pi}	T_{Ri}	T_{Acki}
100	51	2.5	4
110	64	5	6.5
120	62	5	6
180	44	2.5	3.75
150	79	8	6.5

As we can see in Figure 3, by first comparing T_i (m) of RSDC algorithm and T_i (m) of PPDD algorithm that is calculated in Table 3, processing time in RSDC algorithm is optimized.

Table 3. First results for processing time in two algorithms RSDC and PPDD

RSDC T_i(m)	PPDD T_i(m)
6779.7661	7606.0083
6458.3766	7606.0083
6077.3134	7606.0083
6658.1218	7606.0083
6103.5624	7606.0083

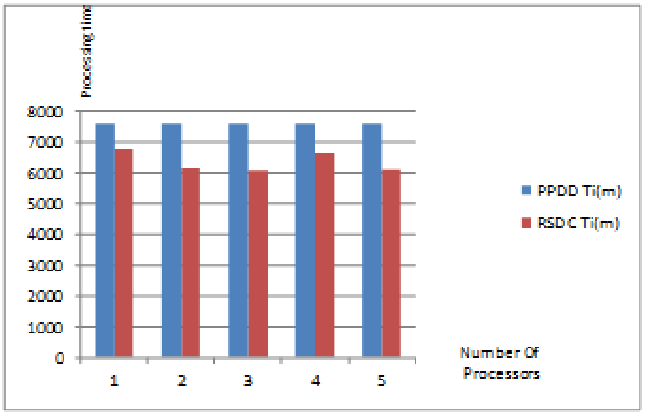

Figure 3. First comparing processing time in two algorithms RSDC and PPDD

We compare the processing time of RSDC algorithm with PPDD algorithm in other dataset that is shown in Table 4. As we can see in Figure 4, by second comparing T_i (m) of RSDC algorithm and T_i (m) of PPDD algorithm that is calculated in Table 5, processing time in RSDC algorithm is optimized.

Table 4. Second dataset using in RSDC and PPDD algorithms

H_i	T_{Pi}	T_{Ri}	T_{Acki}
100	50	2.5	4
110	65	5	6.5
120	60	5	6
180	45	2.5	3.75
150	80	8	6.5

Table 5. Second Results for processing time in two algorithms RSDC and PPDD

RSDC $T_i(m)$	PPDD $T_i(m)$
6816.7377	7608.2993
6124.8685	7608.2993
6153.6350	7608.2993
6617.0636	7608.2993
6076.2065	7608.2993

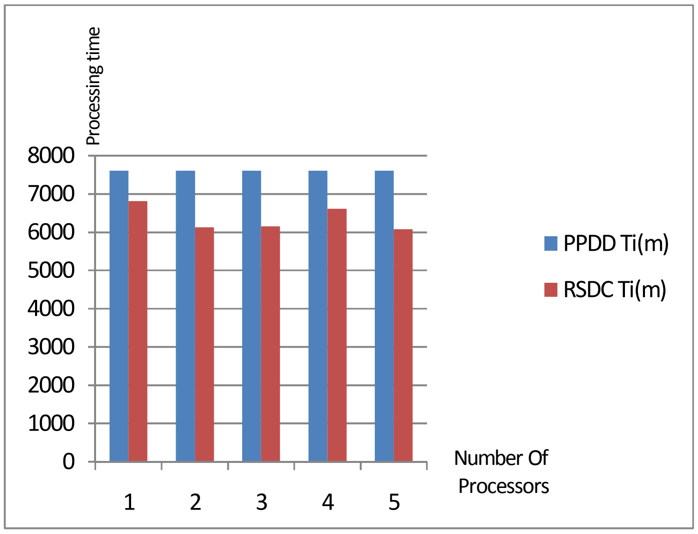

Figure 4. Second comparing processing time in two algorithms RSDC and PPDD

We compare the processing time of RSDC algorithm with PPDD algorithm in other dataset that is shown in Table 6. As we can see in Figure 5, by third comparing T_i (m) of RSDC algorithm and T_i (m) of PPDD algorithm that is calculated in Table 7, processing time in RSDC algorithm is optimized.

Table 6. Third dataset using in RSDC and PPDD algorithms

H_i	T_{Pi}	T_{Ri}	T_{Acki}
100	52	2.5	4
110	66	5	6.5
120	63	5	6
180	42	2.5	3.75
150	81	8	6.5

Table 7. Third results for processing time in two algorithms RSDC and PPDD

RSDC $T_i(m)$	PPDD $T_i(m)$
6871.2561	7637.9603
6212.6288	7637.9603
6208.0074	7637.9603
6545.7488	7637.9603
6151.7528	7637.9603

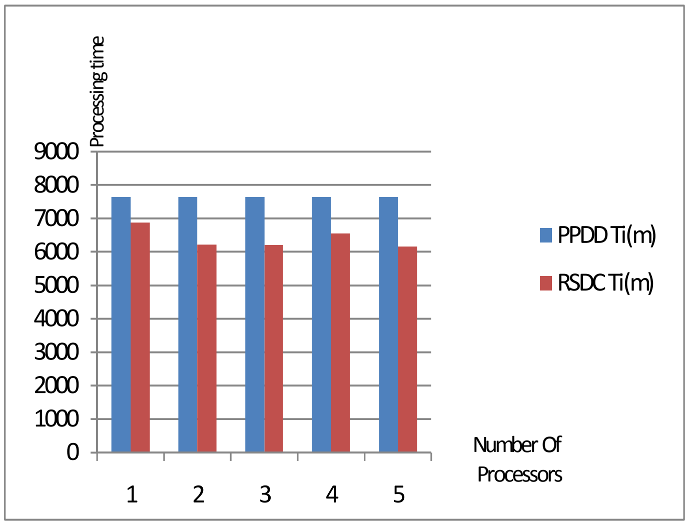

Figure 5. Third comparing processing time in two algorithms RSDC and PPDD

In Table 8 we can see the processing and request and acknowledge time and the initial load for three groups of quintuplet of processors.

Table 8. Data set of three groups of different processors

H_i	T_{Pi}	T_{Ri}	T_{Acki}
100	50	2.5	4
110	65	5	6.5
120	60	5	6
180	45	2.5	3.75
150	80	8	6.5
100	51	2.5	4
110	64	5	6.5
120	62	5	6
180	44	2.5	3.75
150	79	8	6.5
100	52	2.5	4
110	66	5	6.5
120	63	5	6
180	42	2.5	3.75
150	81	8	6.5

Figure 6 shows the way ΔH_i place in RSDC algorithm with the data set shows in Table 8. As can be understood from the diagram the fluctuation rate of ΔH_i in RSDC algorithm is low.

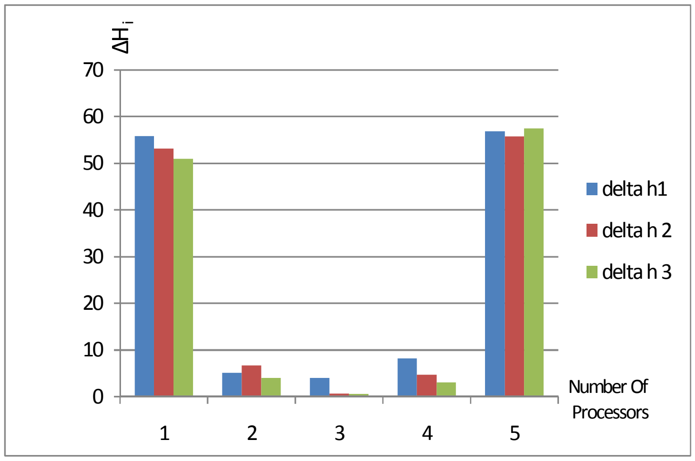

Figure 7 shows the way ΔH_i place in PPDD algorithm with the data set shows in Table 8. As can be seen in the diagram, the fluctuation rate of ΔH_i in PPDD algorithm is higher than the fluctuation rate of ΔH_i in RSDC algorithm.

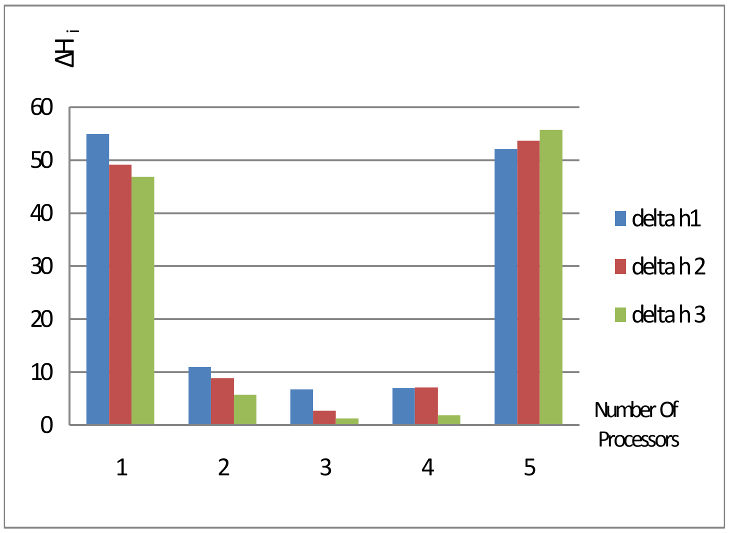

Figure 7. The fluctuation rate of ΔH_i in PPDD algorithm

As can be seen in figures 6 and 7, fluctuation rate of ΔH_i in RSDC algorithm is much less than fluctuation rate of ΔH_i in PPDD algorithm. Decreasing the fluctuation rate of ΔH_i means that the loads which should be subtracted from the initial loads of each processor in S_{over} set or should be added to the initial loads of each processor in S_{under} set do not fluctuated more. It means while adding or subtracting loads, algorithm does not fluctuate more. So algorithm calculates the total processing time with balanced rate. In fact, despite of adding request and acknowledge time to the processing time, algorithm has been successful in computing the average loads for each processor.

According to the diagrams it is obvious that the fluctuation rate of ΔH_i in RSDC algorithm is much lower than the fluctuation rate of ΔH_i in PPDD algorithm. So RSDC is much more efficient than PPDD algorithm.

5. Conclusions

The calculated time is obtained from Table 2 for PPDD algorithm is 7606.01 and for RSDC algorithm is 6779.7661. According to this fact that in PPDD algorithm the processing time is considered without calculating request and acknowledges time, they have been hidden in the processing time. So in RSDC algorithm we subtract the request and acknowledge time from the ultimate time for each processor. And the time that is obtained in RSDC algorithm is much better than the time is obtained in PPDD algorithm.

References

[1] Xiaolin Li · Bharadwaj Veeravalli, (2009) “PPDD: scheduling multi-site divisible loads in single-level tree networks”, www.springer.com, Accepted: 1 September

[2] Arash Ghorbannia Delavar, Behrouz Noori Lahrood, Mohsen Nejadkheirallah and Mehdi Zekriyapanah Gashti, (2011) “ERDQ: A Real-time Framework for Increasing Quality of Customer Whit Data Mining Mechanisms”, International Journal of Information and Education Technology, Vol. 1, No. 1, April

[3] Arash Ghorbannia Delavar, Mehdi Zekriyapanah Gashti, Behroz nori Lohrasbi, Mohsen Nejadkheirallah, (2011) “RMSD: An optimal algorithm for distributed systems resource whit data mining mechanism”, Canadian Journal on Artificial Intelligence, February

[4] Qi Zhang · Lu Cheng · Raouf Boutaba, (2010) “Cloud computing: state-of-the-art and research challenges”, www.springer.com, Accepted: 25 February

[5] Hong-Linh Truong, Schahram Dustdar, (2010) “Composable cost estimation and monitoring for computational applications in cloud computing environments”, www.elsevier.com procedia , International Conference on Computational Science, ICCS

[6] Arash Ghorbannia Delavar, Vahe Aghazarian, Sanaz Litkouhi and Mohsen Khajeh naeini, (2011) “A Scheduling Algorithm for Increasing the Quality of the Distributed Systems by using Genetic Algorithm” , International Journal of Information and Education Technology, Vol. 1, No. 1, April

[7] A. A. H. Liu and A. Hassanien, (2009) “Scheduling jobs on computational grids using fuzzy particle swarm algorithm”, Future Generation Computing Systems.

[8] W A. Abraham, H. Liu and T. Chang, (2008) “Job scheduling on computational grids using fuzzy particle swarm algorithm”, In 10th International Conference on Knowledge Based and Intelligent

[9] Rouhollah Maghsoudi, Arash Ghorbannia Delavar, Somayye Hoseyny, Rahmatollah Asgari, Yaghub Heidari, (2011) “Representing the New Model for Improving K-Means Clustering Algorithm based on Genetic Algorithm”, The Journal of Mathematics and Computer Science Vol .2 No.2 329-336

[10] Arash Ghorbannia Delavar, Nasim Anisi, Majid Feizollahi,Sanaz Litkouhi, (2011) “DLCM: Novel framework of customization improvement in distributed system under data mining mechanism”, Canadian Journal on Data, Information and Knowledge Engineering Vol. 2, No. 2, March