A HYBRID MIN-MIN MAX-MIN ALGORITHM
WITH IMPROVED PERFORMANCE

Kobra Etminani*, Mahmoud Naghibzadeh*, **Noorali Raeiji Yanehsari
*Department of Computer Engineering, Ferdowsi University of Mashad, Iran
**Department of IT, Iran Khodro Khorasan, Iran
etminani@wali.um.ac.ir, naghibzadeh@um.ac.ir, raeiji@ikkco.ir

ABSTRACT
The high cost of supercomputers and the need for large-scale computational resources has led to the development of network of computational resources known as Grid. To better use tremendous capabilities of this large scale distributed system, effective and efficient scheduling algorithms are needed. Many such algorithms have been designed and implemented. We introduce a new scheduling algorithm based on two conventional scheduling algorithms, Min-Min and Max-Min, to use their cons and at the same time, overcome their pros. This heuristic scheduling algorithm, called min-min min-max selective, is evaluated using a grid simulator called GridSim by comparing to its performance against the two basic heuristics which it comes from. The simulation results show that the new heuristic can lead to significant performance gain for a variety of scenarios.

Keywords: task scheduling, grid computing, Min-Min, Max-Min.

1 INTRODUCTION

Today, the increase in supercomputer costs in one hand and the need for large-scale computational resources on the other hand, has led to the development of network of computational resources to solve large-scale problems in science, engineering and commerce. Lots of efforts have been made and many projects have been carried out, such as Globus[1] and Condor[2] to provide the needed concepts and tools for tackling shortcomings. Result of these efforts has led to the emergence of a new paradigm known as Grid.

According to [3], Grid is “a type of parallel and distributed system that enables the sharing, selection and aggregation of geographically distributed autonomous and heterogeneous resources dynamically at runtime depending on their availability, capability, performance, cost and users’ quality of service requirements ….”

One type of grid is computational, that is used for solving large-scale computational problems. A computational grid is “a hardware and software infrastructure that provides dependable, consistent, pervasive and inexpensive access to high-end computational capability” [4].

To make use of great capabilities of this distributed system, effective and efficient scheduling algorithms are needed. Depending on their goals, these algorithms assign tasks to the best machines which produced better quality of service.

Scheduling on a grid has three main phases [5]. Phase one is resource discovery [6], which generates a list of potential resources. Phase two involves gathering information about those resources and choosing the best set to match the application requirements. In phase three the job is executed.

In this paper, we have designed a scheduling algorithm based on two basic scheduling algorithms Min-Min and Max-Min, to use their advantages and at the same time, overcome their disadvantages.

This paper is organized as follows. In section 2, the related works are discussed. In section 3, our scheduling algorithm is introduced. In section 4, the experimental results are presented and discussed. We conclude this study in section 5.

2 RELATED WORK

Many scheduling algorithms [7, 8] have been designed for grid environments, to solve the problem of mapping a set of tasks to a set of machines (scheduling). It has been proved that optimal-solving of this mapping is an NP problem [9]. Many heuristics have been proposed to obtain semi-optimal match. Existing scheduling heuristics can be divided into two categories: on-line mode and batch-mode.

In the on-line mode, a task is mapped to a machine as soon as it arrives at the scheduler. Some heuristic instances of this category follow.

In all following, $m$ denotes number of machines and $s$ denotes number of tasks in a meta-task.

- **MET (Minimum Execution Time):** MET assigns each task to the resource that performs it in the least amount of execution time, no matter whether this resource is available or not at that
time. This heuristic can cause a severe load imbalance across the resources. However, this is one of the heuristics that is implemented in SmartNet [10]. It takes $O(m)$ time to map a given task to an expected resource.

- **MCT (Minimum Completion Time):** MCT assigns each task to the resource which obtains earliest completion time for that task. This causes some tasks to be assigned to resources that do not have minimum execution time for them. This heuristic is also implemented in SmartNet [10]. It takes $O(m)$ time to map a given task to expected resource, too.

- **OLB (opportunistic load balancing):** OLB assigns each task to the resource that becomes ready next, without considering the execution time of the task on that resource. When more than one resource becomes ready, one resource is arbitrarily chosen. The time complexity of OLB is dependent on the implementation. In the implementation considered in [7], it takes $O(m)$ time to find the assignment.

In the batch-mode heuristics, tasks are collected into a set called meta-task (MT). These sets are mapped at prescheduled times called mapping events. Some instances of this category are as follows:

- **Min-Min:** Min-Min begins with the set MT of all unassigned tasks. As shown in Fig. 1, firstly it computes minimum completion time $CT_i$ for all tasks in MT on all resources (lines 1-3). Then two main phases of this algorithm begins. In the first phase, the set of minimum expected completion time for each task in MT is found (lines 5-6). In the second phase, the task with the overall minimum expected completion time from MT is chosen and assigned to the corresponding resource (lines 7-8). Then this task is removed from MT and the process is repeated until all tasks in the MT are mapped (lines 9-11). It is also one of the scheduling algorithms implemented in [10]. This heuristic takes $O(sm)$ time.

- **Max-Min:** Max-Min is very similar to Min-Min, except in phase 2. Max-Min assigns task with maximum expected completion time to the corresponding resource, in phase 2; i.e. replace the underlined word in Fig. 1 minimum with maximum. So, it takes $O(s'm)$ time, too. It is also one of the heuristics implemented in SmartNet [10].

- **Suffrage:** Suffrage [7] is based on the idea that a task should be assigned to a certain resource and if it does not go to that resource, the most it will suffer. Suffrage value for each task is defined through the following equation:

$$ SV = \text{secondMCT} - \text{MCT} $$

where $SV$ is the suffrage value and $MCT$ denotes the minimum completion time and $\text{secondMCT}$ denotes the second minimum completion time. Tasks with high suffrage values take precedence. Suffrage heuristic takes $O(ws)$, where $1 \leq w \leq s$. It can be recognized that in the worst case $w$ is equal to $s$ (i.e. $O(s'm)$) and in the best case $w$ is equal to 1 (i.e. $O(sm)$).

- **XSuffrage:** In Suffrage heuristic, when there is input and output data for the tasks and resources are clustered, Suffrage may have problems described in [11]. To fix the problem, Casanova et al gave an improvement to Suffrage heuristic and created a new heuristic called XSuffrage in [11].

- **QoS Guided Min-Min:** Min-Min heuristic does not consider QoS, which effects its effectiveness in a grid. QoS Guided Min-Min shown in [12] adds a QoS constraint (QoS for a network by its bandwidth) to basic Min-Min heuristic. Its basic idea is that some tasks may require high network bandwidth, whereas others can be satisfied with low network bandwidth, so it assigns tasks with high QoS request first according to Min-Min heuristic. In the worst case, where all tasks need either low QoS or high QoS, this heuristic will take $O(s'm)$ time.

- **Segmented Min-Min:** In Segmented Min-Min heuristic described in [13] tasks are first ordered by their expected completion times. Then the ordered sequence is segmented and finally it applies Min-Min to these segments. This heuristic works better than Min-Min when length of tasks are dramatically different by giving a chance to longer tasks to be executed earlier than where the original Min-Min is adopted.

**Figure 1:** Min-Min Algorithm

```
(1) for all tasks $t_i$ in MT
(2) for all machines $m_j$
(3) $CT_{ij} = ET_{ij} + r_j$
(4) do until all tasks in MT are mapped
(5) for each task $t_i$ in MT
(6) Find minimum $CT_{ij}$ and resource that obtains it.
(7) find the task $t_k$ with the minimum $CT_{kj}$.
(8) Assign $t_k$ to resource $m_i$ that
(9) Delete $t_k$ from MT.
(10) Update $r_i$.
(11) Update $CT_{ij}$ for all $i$.
(12) End do
```
3 SELECTIVE ALGORITHM

Reviewing Min-Min and Max-Min heuristics, it can be seen that depending on the length of unassigned tasks in MT (meta-task), one of these heuristics has better results than the other one [7]. For example, if there is only one long task and too many short tasks, Max-Min will execute long task first and allows short tasks to be executed concurrently with the long task, resulting better makespan and even better resource utilization rate and load balancing level, compared to Min-Min that executes all short tasks first and then executes the long task.

In Table 1, we have shown a sample in which Max-Min outperforms Min-Min. It shows expected execution time of four tasks \((t_x, t_y, t_z, t_w)\) on two machines \((m_x, m_y)\). The machines are assumed to be idle at the start.

Table 1: An example where Max-Min outperforms Min-Min.

<table>
<thead>
<tr>
<th>Task</th>
<th>(m_x)</th>
<th>(m_y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_x)</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>(t_y)</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>(t_z)</td>
<td>4</td>
<td>10</td>
</tr>
<tr>
<td>(t_w)</td>
<td>30</td>
<td>70</td>
</tr>
</tbody>
</table>

As you see in Fig. 2.a, Min-Min gives a makespan of 39, but in Fig. 2.b, Max-Min gives a makespan of 30. Also, in Max-Min, two machines had been working throughout this assignment, but in the Min-Min, the machine \(m_x\) that obtains better completion time, is busy all the time but \(m_y\) is free. So here, Max-Min has better makespan and load balancing level than Min-Min.

On the other hand, in Table 2 we have shown an example in which Min-Min outperforms Max-Min. Similarly there are four tasks and two machines that are assumed to be idle at the start. As it can be seen in Fig. 3.a and Fig. 3.b, Min-Min gave a better makespan than Max-Min.

Table 2: An example where Min-Min outperforms Max-Min.

<table>
<thead>
<tr>
<th>Task</th>
<th>(m_x)</th>
<th>(m_y)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(t_x)</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>(t_y)</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>(t_z)</td>
<td>3</td>
<td>6</td>
</tr>
<tr>
<td>(t_w)</td>
<td>3</td>
<td>6</td>
</tr>
</tbody>
</table>

3.1 Selective Algorithm

Considering advantages and disadvantages of Min-Min and Max-Min, we decided to design a new scheduling algorithm called Selective algorithm to select in each decision, the best of the two algorithms Min-Min and Max-Min, according to length of tasks in a MT in each cycle of task scheduling, as follows:

In the algorithm presented in Fig. 4, all tasks will be sorted according to their minimum execution length, Line 1.
In the first for loop, the algorithm calculates the expected completion time of each task on all resources. Expected completion time of a task on a resource can be calculated using the following relation:

\[ CT_{ij} = ET_{ij} + r_j \]  
(2)

where \( ET_{ij} \) is the expected execution time of task \( t_i \) on machine \( m_j \) and \( r_j \) is the ready time of \( m_j \) i.e. the time when \( m_j \) becomes ready to execute \( t_i \), lines 3 to 5 in Fig. 4.

![Figure 4: Selective Algorithm](image)

In the second for loop, similar to the first phase of Min-Min and Max-Min, it finds minimum expected completion time (such that the task \( t_i \) has earliest expected completion time on machine \( m_j \) ) of each task in MT, and the resource that obtains it, lines 6 and 7 in Fig. 4. If there is more than one resource that obtains this minimum, we choose the resource that has the least resource utilization rate till now, to obtain better load balancing level, lines 8 and 9. According to [14] resource utilization rate can be calculated by using the following relation:

\[ ru_j = \frac{\sum (te_i - ts_i) \text{ where } t_i \text{ has been executed on } m_j}{T} \]  
(3)

where \( te_i \) is the end time of executing \( t_i \) on resource \( m_j \), \( ts_i \) is the start time of executing \( t_i \) on resource \( m_j \) and \( T \) is the total application execution time so far which can be calculated through the following relation:

\[ T = \max(te_j - ts_j) \]  
(4)

where \( t_j \) is executed till now).

Now, to select between Min-Min and Max-Min, a new heuristic is used as follows:

Initially, in line 10, the standard deviation (sd) of \( CT_{ij} \) of all unassigned tasks in MT are calculated, through the following relation:

\[ aveCT = \frac{\sum_{i=1}^{s} CT_{ij}}{s} \]

\[ sd = \sqrt{\frac{\sum_{i=1}^{s} (CT_{ij} - aveCT)^2}{s}} \]  
(5)

where \( aveCT \) denotes average of \( CT_{ij} \).

Relation (5) can be converted to another simple relation as follows:

\[ sd = \sqrt{E(CT_{ij}^2) - E(CT_{ij})^2} \]  
(6)

where \( E(x_i) \) denotes the average of \( x_i \).

Then, a place is found in the sorted list where the difference between the two consecutive values is more than \( sd \), Line 11. By applying this heuristic, the place where a big increase in the length of tasks had occurred, is found. Three cases might happen:

1. If this place is in the first half of the list, it shows that the number of long tasks is more than the number of short tasks i.e. the case where Min-Min outperforms Max-Min. So, we will select Min-Min heuristic, Line 13, to assign the next task.

2. If this place is in the second half, it means that there exists a few long tasks along with too many short tasks i.e. the case where Max-Min outperforms Min-Min. So, we will select Max-Min heuristic, Line 15.

3. If this place does not exist i.e. difference of the two consequence is not more than \( sd \), second condition of Line 12, another heuristic will be used for our selection:

   a. If \( sd \) is less than a certain threshold, it means the length of all tasks are in a small range, so we will select Min-Min to assign the next task.

   b. Otherwise, we will select Max-Min to assign the next task.

After assignment of a task to a corresponding resource, this task will be deleted from MT, Line 16, and the process will be repeated until all tasks will be assigned.

Fig. 5 gives an example that shows how
selective algorithm selects between Min-Min and Max-Min. Assume that minimum $CT_i^j$ for each task is found and listed in Fig. 5 and $sd$ is calculated, too (i.e., lines 1 to 10 of Fig. 4 is executed).

In Fig. 5.a, there exists one short task and five long tasks, the case where Min-Min outperforms Max-Min. As it can be seen, occurrence of the place of difference is in the first half, so Min-Min is selected to assign next task. But in Fig. 5.b, there are five short tasks and one long task, the case where Max-Min outperforms Min-Min. Therefore, this place is in the second half, so this algorithm applies Max-Min.

![Figure 5: Example of selection in Selective Algorithm](image)

### 3.2 Time Complexity of Selective Algorithm

To compare the proposed algorithm with its two basic heuristics in time complexity measure, we computed the time complexity of Selective algorithm here.

In Line 1 of Fig. 4, an array of length $s$ is being sorted. By using any sorting algorithm, it takes $O(s^2)$ time in its worst case.

In lines 3-5, two nested for loops take $O(s.m)$ time: internal for loop runs $m$ times (number of machines) and external for loop runs $s$ times (number of tasks).

Finding minimum $CT_i^j$ takes $O(m)$ time (line 7).

Finding the machine with minimum resource utilization rate in worst case, when all machines have same $CT_i^j$, takes $O(m)$ time. This is done for all tasks (lines 7-9), so it takes $O(s.m)$ time.

Computing standard deviation (line 10) consisted of calculating the average of $s$ numbers: average of the array of $CT_i^j$ and average of the array of $(CT_i^j)^2$ both with $s$ members. So it takes $O(s)$ time.

Finding the place $p$ in a list with $s$ members (line 11) needs $O(s)$ time. It is a sequential search.

Selection part of the new algorithm (lines 12-15) takes $O(1)$ time, because the list is sorted and one should go to the start (for Min-Min) or the end (for Max-Min) of the list and no need to find minimum or maximum.

Deleting the assigned task, Line 16, takes $O(1)$, too, because the list is sorted and the task is deleted from the start or the end.

Therefore, time complexity of lines 3-16 is the maximum of $O(sm)$, $O(sm)$, $O(s)$, $O(s)$, $O(1)$, and $O(1)$, that is $O(sm)$.

This process, lines 3-16, is done for all tasks in $MT$; i.e., runs $s$ times. Therefore, lines 2-17 takes $O(s^2m)$ time.

Consequently, time complexity of the Selective algorithm is:

$$\max(O(s^2), O(s^2m)) = O(s^2m)$$

Comparing it to Max-Min and Min-Min, the new heuristic does not impose any extra load and has the same time complexity as them.

### 4 Experimental Results

#### 4.1 Performance Metrics

Depending on what scheduling performance is desired in grid there exist different performance metrics for evaluating these algorithms. Some of these metrics are introduced here.

- **Makespan**: Makespan is a measure of the throughput of the heterogeneous computing systems, such as grid. It can be calculated as the following relation:
  $$\text{makespan} = \max_i (CT_i^j)$$

The less the makespan of a scheduling algorithm, the better it works.

- **Average resource utilization rate**: It is one of the metrics that is used in [14]. Average resource utilization of each resource can be calculated through relation (2). Average resource utilization of total resources is calculated through the following relation:
  $$ru = \frac{\sum_{j=1}^{m} ru_j}{m}$$

where $ru$ is in the range 0 to 1.

- **Load balancing level**: The mean square deviation of $ru$ is defined as:
  $$d = \frac{\sum_{j=1}^{m} (ru_j - ru)^2}{m}$$

and the relative deviation of $d$ over $ru$ that determines load balancing level is:
  $$\beta = \left(1 - \frac{d}{ru}\right) \times 100\%$$

The best and most efficient load balancing level is achieved if $d$ equals zero and $\beta$ equals 1. So, scheduling algorithm will have better performance if $d$ is close to 0 and $\beta$ is close to 1. It is the other metric that is used in [14].
4.2 Experiments

4.2.1 Simulation Environment

To evaluate and compare our scheduling algorithm with its two basic heuristics Min-Min and Max-Min, a simulation environments known as GridSim toolkit [15] had been used. There are several grid simulators that allow evaluating a new grid scheduling algorithm, such as Bricks [16], MicroGrid [17] and SimGrid [18]. But GridSim has some good advantages which are listed below:

- It allows modeling of heterogeneous types of resources.
- Resource capability can be defined (in the form of MIPS (Million Instructions Per Second) as per SPEC (Standard Performance Evaluation Corporation) benchmark).
- There is no limit on the number of application jobs that can be submitted to a resource.
- It supports simulation of both static and dynamic schedulers.

GridSim had been used in many researches to evaluate the results, such as [19, 20, 21].

4.2.2 Experimental Data and Results

The experimental testing of our heuristic is performed in three scenarios:

1. Scenario I: A few short tasks along with many long tasks; i.e. the case where Min-Min outperforms Max-Min.
2. Scenario II: A few long tasks along with many short tasks; i.e. the case where Max-Min outperforms Min-Min.
3. Scenario III: A few long tasks along with many short tasks; i.e. the case where Max-Min outperforms Min-Min.

Number of resources is chosen to be 10. Three different numbers of tasks has been chosen: 500, 1000 and 2000, to be sure of efficiency of the proposed heuristic. The task arrivals are modeled by a Poisson random process.

Result of this simulation is as follows:

In Fig. 6, Fig. 7 and Fig. 8, it can be seen that the Selective algorithm tries to do its best in each scenario, with 500, 1000 and 2000 tasks respectively. In scenario 1 (Fig. 6.a, Fig. 7.a and Fig. 8.a), which Min-Min outperforms Max-Min, it has the makespan the same as Min-Min.

In scenario 2 (Fig. 6.b, Fig. 7.b and Fig. 8.b), which Max-Min outperforms Min-Min, it acts like Max-Min and in scenario 3 (Fig. 6.c, Fig. 7.c and Fig. 8.c), it tends to have makespan the same as the best algorithm, here it was Min-Min. Increasing the number of tasks, better efficiency can be observed, too.

In Fig. 9, Fig. 10 and Fig. 11, which show the average resource utilization rate for 500, 1000 and 2000 tasks respectively, you can see that, again, Selective heuristic performs like the best heuristic in each scenario. Even, in the third scenario, that is more similar to the real cases, it acts better than the two basics.
Figure 9: average utilization rate for 500 tasks

In Fig. 12, Fig. 13 and Fig. 14, in every three scenarios for 500, 1000 and 2000 tasks respectively, Selective algorithm acts like the best algorithm. Here, in load balancing level metric, Max-Min has better load balancing level than Min-Min because, as explained in section 3, Min-Min assigns the task with the earliest completion time in each phase, results in some resources becoming busy all the time and others becoming free most of the time. Therefore, it has less load balancing level than Max-Min where it assigns the task with maximum completion time and lets other tasks executes along on the other resources, therefore have better load balancing level.

Figure 10: average utilization rate for 1000 tasks

Figure 11: average utilization rate for 2000 tasks

Figure 12: load balancing level for 500 tasks

Figure 13: load balancing level for 1000 tasks

Figure 14: load balancing level for 2000 tasks

5 Conclusion and Future Work

To achieve high computing throughput in a grid environment, this new scheduling algorithm was proposed. It selects between two conventional algorithms, Min-Min and Max-Min, whenever one acts better than the other based on the standard deviation of minimum completion time of all unassigned tasks in a meta-task. Evaluation of our new heuristic was done through a simulation environment called GridSim. The experimental results show that the Selective algorithm outperforms the traditional Min-Min and Max-Min heuristics.

This study is concentrated only on standard deviation. Many similar heuristics can be devised. Many issues remain open. We did not consider deadline of each task, cost of execution on each resource, cost of communication and many other cases that can be topics of further research. Finally, we intend to reuse our new scheduling heuristic in an actual environment for practical evaluation.

6 REFERENCES


Kobra Etminani is a PH.D. student in the Department of computer engineering at Ferdowsi University of Mashad. She received the M.S. degree in computer engineering from Ferdowsi University of Mashad in 2007 and the B.S. degree in computer engineering from that university in 2005. Her research interests include grid computing, data mining and knowledge management.

Mahmoud Naghibzadeh received his B.S. degree in Statistics and Computer Science from Ferdowsi University of Mashad in 1975, M.S. degree in Computer Science and Ph.D. degree in Electrical Engineering –Computer from University of Southern California (USC). He is a Full Professor of Computer Engineering at Ferdowsi University of Mashhad.
now. His research interests include Operating system design concepts, especially scheduling real-time processes, Distributed operating systems designs concepts, Distributed databases design concepts and knowledge management. He has published 9 books, over 100 journal and conference papers in these areas.

Noorali Raeji Yanehsari is working in Network Administration Unit of Iran Khodro Khorasan (a branch of the largest car factory in Iran). He received his B.S. degree in computer engineering from Ferdowsi University of Mashad in 2003. His research interests include network security, knowledge management and grid computing.