Two Stages Transfer Algorithm (TSTT) for Independent Tasks Scheduling in Heterogeneous Computing Systems

1Abdulrahman K. Al-Qadhi, 1Ahmad Alauddin Ariffin, 1,3 Rohaya Latip, 1Nor Asila Wati Abdul Hamid, 2Ammar S. Al-Zubaidi

1Universiti Putra Malaysia, Faculty of Computer Science and Information Technology, Selangor, Malaysia
2University of Baghdad, Computer Centre, Baghdad, Iraq
3Institut Penyelidikan Matematik (INSPEM), Universiti Putra Malaysia, Selangor, Malaysia
E-mail: prog.abdulrahman@gmail.com

Abstract. Task scheduling is critical in heterogeneous systems especially with the huge number of tasks transmitted over grid causing system delay. Since heuristics are proposing methods for solving heterogeneous computing systems, several techniques were proposed for the scheduling on grid computing systems to get better execution time. In this paper, a proposed new heuristic algorithm named Two Stages Tasks Transfer (TSTT) algorithm to enhance Tenacious Penalty Based scheduling (TPB) algorithm. Heterogeneous Computing Scheduling Problem (HCSP) mathematical model has been used, where the independent tasks assigned to heterogeneous machines with different characteristics. Twelve datasets with different heterogeneity level examined using different heuristic algorithms to compare the performance with our proposed algorithm. The proposed algorithm showed its efficiency in term of makespan, resource utilization metrics for set of tasks.

1. Introduction
In the last two decades, the high speed low-cost computers production led to boost the processing power through networks; this fact helps to grow in distribution of distributed systems to contain and tap these resources [1,2].

A heterogeneous computing (HC) referring to set of heterogeneous elements called computers, resources, processors or machines, these resources connected to each other by a network [2]. Figure 1 shows the heterogeneous computing environment.

Cloud computing and Grid computing are famous examples of distributed computing resources working over loosely coupled virtual machines, created by connecting many heterogeneous computing resources with different characteristics. Where both cloud and infrastructures provide an easy low cost solutions to access for distributed computing resources distributed over the globe to solve large problems [3,4].
The main issue when using a distributed computing systems environment is finding an efficient way for task scheduling to satisfy the user needs of fast computing power, where tasks should be assigned to the resources in a clever way [2].

In this research, we aim to propose an efficient solution for Heterogeneous Computing Problem (HCSP) and evaluate this method to ensure its efficiency.

2. Related Works
Tasks and machines in general are heterogeneous with different capabilities and different properties. The large scale of resource and big number of tasks can lead to system degrading; this fact motivated the researcher to do many researches on scheduling methods for heterogeneous computing systems. Resource heterogeneity, tasks heterogeneity and task dependencies were the main issues to deal with during conducting the research [5].

HCSP attracted many researchers, where many techniques used to solve this problem such as heuristic and metaheuristic techniques [5,6].

Min-Min algorithm introduced by [7] and [8]. Min-Min schedule the task depends on earliest finishing time (EFT) value for each machine. Despite its good results, scheduling in Min-Min seems bad idea when the number of short tasks is much higher than the number of long tasks where there is no many options for scheduling [9,10].

Authors in [11] and [12] proposed load balancing algorithm. Min-mean algorithm derived from Min-Min algorithm. Where this algorithm has two steps, in first step, Min-Min algorithm was used. In the second step, mean execution time (meanCT) computed and compared with all machines’ finish time, if the machine with EFT less than meanCT, then the task with maximum EFT value will be scheduled, otherwise, the task with minimum EFT will be scheduled. The algorithm showed better results comparing to min-min algorithm in term of total execution time. But the problem of having majority of loaded machines was not solved because the algorithm depends on Min-Min algorithm in its first stage.

Unlike Min-Min which select depending on earliest finishing time, the MaxMin in [6,8,13] choose tasks to be scheduled depending on maximum EFT value[14]. Another problem appeared when the number of long tasks is much higher than the short tasks, where it might cause makespan to degrade less than Min-Min [10].
An improved algorithm Max-Average was developed by [15] to improve Max-Min algorithms by calculating the average completion time for all machines then assign the maximum completion time task to minimum estimated completion time resource. The new algorithm shows a better result comparing to Max-Min algorithm in term of makespan.

Sufferage discussed by [13] and [8], where the tasks in Sufferage algorithm might suffer if the task does not schedule to the machine which can execute it faster. The suffer value compute by finding the difference between the machine with earliest finish time (EFT) and the machine with second earliest finish time. After calculating the suffer value for all tasks, the task with highest suffer value will be assigned to the machine with lowest completion time. The main drawback of this algorithm is calculating the suffer value in each iteration which cause more time to be lost.

LSufferage proposed by [1] is an enhancement of Sufferage algorithm trying to overcome its drawback by adding the priority list to store the priority of execution for each task, where each machine has a list of priorities. The priorities of tasks which run on faster machine are higher than other machines. The results of this algorithm didn’t show better results in term of makespan comparing to other heuristics in the paper.

A heuristic algorithm with load balancing mechanism named penalty based (PB) were proposed by [16]. The tasks in this algorithm assigned to machine with minimum execution time in first phase. Second phase did the load balancing by moving the task from the most loaded machine to less loaded machine, task penalty in this case must be minimum.

TPB is an enhanced algorithm; this algorithm is an enhancement of PB algorithm proposed by [1]. The idea of moving the task from the most loaded machine to least loaded machine has been extended. Tenacious penalty based algorithm (TPB) move the task from the most loaded machine to any other machine by checking with other machines after sorting them depending on load in ascending order. The authors compared between the new two proposed algorithms (LSufferage and TPB) with some other heuristics. The results showed that their new algorithm TPB is better in term of makespan.

Even TPB showed excellent results in term of makespan, researchers always aim to reach optimality. In our research, we aim to reach an optimal solution to execute tasks faster and improve the utilization.

3. Experiment
Heterogeneous Computing Scheduling Problem (HCSP) become very important issue, especially with the increasing in use distributed computing with heterogeneous resources [17].

Estimated Tasks Completion (ETC) matrices proposed by [6,18] used to test and evaluate our proposed algorithm. ETC matrix contains the expected execution time for the task on each resource. Each element in matrix ETC[t,p] represents the execution time for task t in resource p, assuming that the transfer time of the task included. These values assume to be known a priori approaches that can be used. This estimated values based on analytical benchmarking and profiling [6]. With twelve datasets proposed by [6] which consider all heterogeneity and consistency for both tasks and machines, these testbeds become as one of main standards benchmarks which can evaluate and solve heterogeneous computing scheduling problem (HCSP). Table 1 below showing the heterogeneity levels for datasets used in our experiments.

| Heterogeneity | Consistency |
|---------------|-------------|
| Task          | Resource    | Consistent | Inconsistent | Semi-consistent |
| High          | High        | u_c_hihi   | u_i_hihi     | u_s_hihi       |
| High          | Low         | u_c_hilo   | u_i_hilo     | u_s_hilo       |
| Low           | High        | u_c_lohi   | u_i_lohi     | u_s_lohi       |
| Low           | Low         | u_c_lolo   | u_i_lolo     | u_s_lolo       |
ETC model considered as one of most important computational models, work to solve scheduling problems as [2,5] mentioned. Full details, datasets and some experimental results can be downloaded from [19].

For this purpose, HCSP mathematical model developed by [1] to deal with the ETC matrices has been used. This model designed to achieve a better understanding for problem, and better scheduling results as well by concerning the makespan value minimization as a main objective.

4. The proposed Algorithm

Our proposed algorithm inspired by TPB algorithm proposed by [1] which has very efficient executing time.

Tenacious penalty based algorithm (TPB) proposed has two phases. In first phase, TPB sort the machines depending on their load in ascending order; assign the tasks to earliest finishing time machine. In the second phase, the tasks move from the most loaded machine to any other machine according to penalty value by checking with other machines after sorting them depending on load in an ascending order. Table 2 below shows the definition of notations used in pseudo code for both Algorithm 1 and Algorithm 2.

| Notation | Definition |
|----------|------------|
| comp_a  | Completion time of pa |
| cb      | Completion time of pb |
| ETC     | Estimated Completion Time |
| m_penalty | Minimum penalty |
| new_cb  | New completion time for the selected machine |
| new-comp_a | new completion time for the Maximum loaded machine |
| new_ms  | New calculated makespan |
| P       | Set of all machines |
| pa      | Most loaded machine |
| pb      | Selected machine |
| pl      | Set of machines sorted in ascending order |
| rep     | Repetitions |
| s_proc  | Selected machine |
| s_task  | Selected task to be transferred from pa |
| s_t_1   | Selected task from pa to be exchanged with s_t_2 from pb |
| s_t_2   | Selected task from pb to be exchanged with s_t_1 from pa |
| T       | Set of all tasks |

Algorithm 1  Tenacious penalty based (TPB)

\[ \forall \ t \ where \ t \in T , Schedule \ t \ to \ the \ machine \ which \ can \ execute \ the \ tasks \ faster \]

\[ rep = 0 \]
while rep < T
    rep = rep + 1
    m_penalty = +∞
    s_task  = −1

    for all pb ∈ pl excluding pa do
        for all t ∈ tasks scheduled in pa do
            if cb +ETC(t, pb) < comp_a then
                penalty = (ETC(t,pb)-ETC(t,pa))/(ETC(t,pa))
                if penalty < m_penalty then
                    m_penalty = penalty
                    s_task  = t
                    s_proc = pb
                end if
            end if
        end for
        if s_task ≠ −1 then
            Transfer s_task from pa to s_proc
            Continue the while loop
        end if
    end for
    if s_task = −1 then
        return
    end if
end while

The need for fast task execution time in distributed computing systems increased rapidly. This need was a motivation to propose an efficient algorithm to reduce the execution time and increase the utilization between the resources. In this section, a proposed algorithm Two Stages Tasks transfer (TSTT) presented. From its name, we can conclude that the proposed algorithm has two stages. The first stage adopted from Tenacious Penalty Based algorithm (TPB) algorithm proposed by [1], where the scheduler tries to reduce the makespan by transferring the task from most loaded processor to any other machine. The task and its receiver machine were chosen according to a penalty equation which can lead to reduce the makespan. The proposed algorithm TSTT extends the idea of TPB by adding another step where tasks can be exchanged between the two chosen machines depending on a comparison value between the estimated makespan of the chosen machine and the maximum loaded machine.

Algorithm 2  Two Stages Tasks Transfer (TSTT)
Stage 1: is equivalent to TPB algorithm (Algorithm 1)
Stage 2: Tasks exchange stage:

For all t1 ∈ T where t1 scheduled in pa
    For all p in P except the pa {
        Selected machine (pb)
        cb  → (selected machine completion time)
        For all tasks t2 ∈ T where t2 scheduled in pb
            new-comp_a  → (new completion time for the Maximum loaded machine)
            new-comp_a= comp_a- ETC(t1, pa)+ETC(t2, pa)
            new_cb = cb − ETC(t2,pb) + ETC(t1,pb)
if (ca_new < comp_a && cb_new < ca) then
    new_ms = max(ca_new, cb_new); // new makespan
    If (new_ms < ca)
        \( s_{t_1} = t1 \)
        \( s_{t_2} = t2 \)
        \( s_{proc} = pb \)
    End if
End if
End for
If tasks selected \( \rightarrow \) transfer \( s_{t_1} \) from \( pa \) to \( s_{proc} \)
    \( \rightarrow \) transfer \( s_{t_2} \) from \( s_{proc} \) to \( pa \)
End if
End for

The pseudo code presented in Algorithm 2 represented the details of the proposed algorithm. By checking if the task exchange between the maximum loaded machine \( pa \) and the selected machine \( pb \) will lead to reduce the makespan, then the scheduler will make a decision to do a task exchange. Otherwise the scheduler will keep searching till find a machine with the task which lead to decrease the makespan. This algorithm set reducing the total execution time (makespan) as a goal.

5. Results and discussion
In this paper, two important performance metrics have been used to show the efficiency of state of the art algorithms in term of makespan and resource utilization. The proposed algorithm will be compared with well-known heuristics like Sufferage, penalty based and Tenacious penalty based algorithms.

5.1. Parameters
The parameter settings performed depending on the twelve problem datasets [19] were used. In this experiment, datasets tested 512 X 16. This indicates 512 tasks and 16 machines. Java based simulation has been used on Intel (R) core(TM) i7 -6500U CPU @2.5 GHz, 2.9 GHz to run this simulation. The research has been done using the parameters shown in table 3 below:

| Parameters used in the implementation |
|---------------------------------------|
| Tasks                                 | 512 |
| Machines                              | 16  |
| Task heterogeneity                    | High, low |
| Machine heterogeneity                 | High, low |
| Machine consistency                   | Consistent, inconsistent, semi-consistent |
| Datasets                              | \( u_c_{hihi}, u_c_{lohi}, u_c_{hilo}, u_c_{lolo}, u_i_{hihi}, u_i_{lohi}, u_i_{hilo}, u_i_{lolo}, u_s_{hihi}, u_s_{lohi}, u_s_{hilo}, u_s_{lolo} \) |

5.2. Performance metrics
In this paper, two performance metrics has been used to show the efficiency of our proposed algorithm in term of makespan, resource utilization. The proposed algorithm compared with well-known heuristics like Sufferage, Penalty Based, and Tenacious penalty based algorithms.

5.2.1. Makespan
Makespan is the time takes the scheduler to schedule all tasks in its suitable resources, where the makespan is the main objective in scheduling process [20]. Makespan equation (1) listed below:

\[ \text{Makespan} = \sum_{i=1}^{n} \text{Task Duration}_i \]

\[ \text{Makespan} = \sum_{i=1}^{n} \frac{\text{Task Duration}_i}{\text{Resource Utilization}_i} \]
Makespan = \( \max \{ \text{completion}[t], \text{where } t \in T \} \) … (1)

In our experiment we used the lower bound (LB) values to represent the optimal execution time to solve the scheduling problem. These values are unrealistic values, because it is assuming that the system can’t be interrupted under any circumstances like some physical or even task preemption. These values calculated using linear programming relaxation assuming every task in HCSP instances always executed on fastest machine (2). The LB values for all the twelve datasets published in [19].

The gap between the optimal solution and the current solution for the heterogeneous computing scheduling problem (GAP) presented in equation (2) were introduced by [21].

\[
\text{GAP} = \frac{\text{Current makespan} - \text{Lower bound value}}{\text{Lower bound value}} \quad (2)
\]

**Figure 2.** The gap between calculated makespan and lower bound for the consistent datasets (u_c_hihi, u_c_loho, u_c_hiho, u_c_lolo).

**Figure 3.** The gap between calculated makespan and lower bound for the inconsistent datasets (u_i_hihi, u_i_lohi, u_i_hiho, u_i_lolo).
Fig. 4. The gap between calculated makespan and lower bound for the semi-consistent datasets (u_s_hihi, u_s_loho, u_s_hilo, u_s_lolo).

The figures 2, 3, and 4 showed graphical summaries of the makespan gap from the optimal solution. The datasets categorized according to the consistency into consistent, inconsistent and semi-consistent. Figure 2 represents the consistence datasets, while figure 3 represents the inconsistent dataset, and figure 4 represents the semi-consistent one.

From the result shown above in figures 2, 3 and 4, we can observe that the bar of (TSTT) has a least value among all heuristics compared with it (Sufferage, PB and TPB). This means that the proposed algorithm is closer to optimality comparing to other examined algorithms.

5.2.2. Resource Utilization

Resource utilization “is one of the main characteristics in heterogeneous computing. Achieving high resource utilization is the goal for researchers to gain benefits from idle resources. We can also say it is a quality of service metric. Resource utilization can be defined as the average utilization of the resources”. Taking full advantage of resources is an important objective in heterogeneous computing scheduling. The importance of the resource utilizations because of the economic effect [5].

\[
\text{Resource utilization} = \frac{\sum_{p \in P} \text{Completion time (p)}}{\text{makespan} \times np} \quad (3)
\]

Where \( p \in P \) (Set of all the machines), and \( np = \) number of machines.
Figure 5. The resource utilization for the tested heuristics using the consistent datasets (u_c_hihi, u_c_loho, u_c_hilo, u_c_lolo).

Figure 6. The resource utilization for the tested heuristics using the inconsistent datasets (u_i_hihi, u_i_lohi, u_i_hilo, u_i_lolo).
Figure 7. The resource utilization for the tested heuristics using the semi-consistent datasets (u_s_hihi, u_s_loho, u_s_hilo, u_s_lolo).

The figures 5, 6 and 7 summarized the resource utilization results of the tested algorithms for all heterogeneity and consistency prospects. The datasets categorized according to the consistency into consistent, inconsistent and semi-consistent. Figure 5 represent the consistence datasets, while figure 6 represent the inconsistent dataset, and figure 7 represent the semi-consistent one. As shown in these figures we can observe that the orange color bar (TSTT) has a better utilization comparing to the other tested heuristics, where the result for the proposed algorithm is closer to the maximum utilization (closest to 100% utilization).

However, these results have not previously been discussed. Figure 2 showed the gap of produced makespan from the optimal makespan in consistent datasets. The results showed that the proposed algorithm always has a better execution time comparing to other heuristics; this might be because the new algorithm have a better load balancing for the reason that the proposed algorithm always make sure to produce better makespan by transferring and exchanging tasks to get better results in term of execution time.

Figures 3 and 4 presented the gap of produced makespan from the optimal makespan in inconsistent and semi-consistent datasets, the results showed that the proposed algorithm always produce better results from other examined heuristics, but in some datasets, such as u_i_hihi, u_i_lohi and u_s_hihi, this algorithm share the best results with TPB algorithm. These results may be explained by the fact that the TSTT is the proposed enhancement of TPB algorithm done by [1], where a new stage has been added to enhance the algorithm and produce better makespan by exchanging the tasks between the resources; This gives a reason where this result might be generated because there are no tasks have been exchanged because the condition of tasks exchange in the second phase have not met.

We can notice that the same datasets (u_i_hihi, u_i_lohi and u_s_hihi) in resource utilization figures (5, 6 and 7); the proposed method TSTT and TPB algorithm got same values, while in other nine datasets produce better values. This might happen as we mentioned previously because of there are no tasks have been exchanged because the condition of task exchange in the second phase have not met, where the results of resource utilization influenced by makespan value [20].

6. Conclusion
The scheduling in heterogeneous computing system is critical problem; especially with the increasing in heterogeneous resources usage in different heterogeneous environments like grid computing and cloud computing with increasing in the tasks submitted to these resources. This fact lead to increases the need to provide an efficient task scheduling method which can be reduce tasks execution time, and
utilization between resources to achieve fast services for the system clients and optimal utilization between resources.

An enhanced algorithm proposed in this paper used a load balancing technique to provide better results in term of makespan and resource utilization. Two stages tasks transfer mechanism used to achieve better load balancing by moving the tasks from most loaded machine to any other machine in a way where make sure that the new makespan always lesser than the previous one. The results showed that the proposed method provide better results in term of makespan, resource utilization than other well-known scheduling algorithms.

For future work, we plan to consider other specification, such as: tasks deadline, and task preemption in our research.

References

[1] Gogos C, Valouxis C, Alefragis P, Goulas G, Voros N, Housos E. 2016 Scheduling independent tasks on heterogeneous processors using heuristics and Column Pricing. *Futur Gener Comput Syst*. **60**:48–66.

[2] Nesmachnow S. 2013 Parallel multiobjective evolutionary algorithms for batch scheduling in heterogeneous computing and grid systems. *Comput Optim Appl*. **55**(2):515–44.

[3] Foster I, Kesselman C. 2004 The grid 2: Blueprint for a new computing infrastructure [Internet]. Morgan Kauffman. p. 748. Available from: http://scholar.google.com/scholar?hl=en&btnG=Search&q=intitle:The+GRID+2%3A4%5Cnhttp://books.google.com/books?hl=en&lr=&id=0l5gm6o3vrMC&oi=fnd&pg=PP2&dq=The+grid+2:+Blueprint+for+a+new+computing+infrastructure&ots=hoHjwTKvrd&sig=KZWHelmc-KqVg7_nkkEpfyhCb6U

[4] Panda SK, Gupta I, Jana PK. 2015 Allocation-aware task scheduling for heterogeneous multi-cloud systems. *Procedia Computer Science*. p. 176–84.

[5] Xhafa F, Abraham A. 2008 Meta-heuristics for grid scheduling problems. *Stud Comput Intell*. **146**:1–37.

[6] Braun TD, Siegel HJ, Beck N, Bölöni LL, Maheswaran M, Reuther AI, et al. 2001 A Comparison of Eleven Static Heuristics for Mapping a Class of Independent Tasks onto Heterogeneous Distributed Computing Systems. *J Parallel Distrib Comput* [Internet]. **61**(6):810–37. Available from: http://dx.doi.org/10.1006/jpdc.2000.1714

[7] Ibarra OH, Kim CE. 1977 Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors. *J ACM*. **24**(2):280–9.

[8] Casanova H, Legrand A, Zagorodnov D, Berman F. 2000 Heuristics for scheduling parameter sweep applications in grid environments. In: *Proceedings 9th Heterogeneous Computing Workshop (HCW 2000)* (Cat NoPR00556) [Internet]. p. 349–63. Available from: http://ieeexplore.ieee.org/lpdocs/epic03/wrapper.htm?arnumber=843757

[9] Anousha S, Ahmadi M. 2013 An improved Min-Min task scheduling algorithm in grid computing. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*. p. 103–13.

[10] Tabak E, Cambazoglu B, Aykanat C. 2013 Improving the Performance of Independent Task Assignment Heuristics Minmin, Maxmin and Sufferage. *IEEE Trans parallel Distrib Syst*. **25**(5):1–14.

[11] Kamalam G, Muralibhaskaran V. 2010 A new heuristic approach: Min-Mean algorithm for scheduling meta-tasks on heterogeneous computing systems. *Int J Comput Sci Netw Secur*. **10**:24–31.

[12] Kamalam G, Bhaskaran V. 2012 New enhanced heuristic min-mean scheduling algorithm for scheduling meta-tasks on heterogeneous grid environment. *Eur J Sci Res*. **70**(3):423–30.

[13] Maheswaran M, Ali S, Siegel HJ, Hensgen D, Freund RF. 1999 Dynamic Mapping of a Class of Independent Tasks onto Heterogeneous Computing Systems. *J Parallel Distrib Comput*. **59**(2):107–31.

[14] Chang RS, Lin CY, Lin CF. 2012 An Adaptive Scoring Job Scheduling algorithm for grid computing. *Inf Sci (Ny)*. **207**:79–89.
[15] Maipan-uku J, Muhammed A, Abdullah A, Hussin M. 2016 Max-Average: An Extended Max-Min Scheduling Algorithm for Grid Computing Environment. *J Telecommun Electron Comput Eng.* 8(6):43–7.

[16] Chaturvedi AK, Sahu R. 2011 New Heuristic for Scheduling of Independent Tasks in Computational Grid. *Int J Grid Distrib Comput.* 4(3):25–36.

[17] Nesmachnow S, Canabé M. 2011 GPU implementations of scheduling heuristics for heterogeneous computing environments. *XVII Congreso Argentino de Ciencias de la Computación.*

[18] Ali S, Siegel HJ, Maheswaran M, Hensgen D. 2000 Task execution time modeling for heterogeneous computing systems. *Proceedings 9th Heterogeneous Computing Workshop (HCW 2000) (Cat NoPR00556).* p. 185–99.

[19] CECAL: HCSP- Heterogeneous Computing Scheduling Problem [Internet]. CECAL. 1993. Available from: https://www.fing.edu.uy/inco/grupos/cecal/hpc/HCSP/

[20] Gogos C, Valouxis C, Alefragis P, Xanthopoulos I, Housos E. 2016 Scheduling independent tasks on heterogeneous computing systems by optimizing various objectives. *Proceedings of the 11th International Conference on Practice and Theory of Automated Timetabling (PATAT-2016).* p. 149–61.

[21] Nesmachnow S, Cancela H, Alba E. 2012 A parallel micro evolutionary algorithm for heterogeneous computing and grid scheduling. *Appl Soft Comput J.* 12(2):626–39.