Journal of Computer Sciences

Original Research Paper

Scheduling Jobs through Gap Filling and Optimization Techniques in Computational Grid

Omar Dakkak, Shahrudin Awang Nor and Suki Arif

InterNetWorks Research Laboratory, School of Computing,
Universiti Utara Malaysia, 06010 UUM Sintok, Kedah, Malaysia

Article history
Received: 13-03-2016
Revised: 22-04-2017
Accepted: 17-05-2017

Corresponding Author:
Omar Dakkak
InterNetWorks Research Laboratory, School of Computing, Universiti Utara Malaysia, 06010 UUM Sintok, Kedah, Malaysia
Email: oaldakkak@gmail.com

Abstract: Due to the heterogeneity and complexity in grid computing, classical algorithms may not be able to deal with dynamic jobs properly. In the dynamic mode, incoming jobs reach the scheduler arbitrary. Therefore, scheduling the jobs using simple policy alone deteriorates the performance of the scheduler. Thus, a policy that can handle the dynamicity efficiently is indispensable. This paper presents the Swift Gap mechanism (SG), which is a hybridization of the Best Gap mechanism, alongside with Tabu search (BGT). In addition, a new decision rule based on completion time is included into the outcome mechanism. The new decision rule based on completion time has shown a significant improvement in the Quality of Service (QoS), especially for a slowdown, tardiness, waiting time and response time. Moreover, an evaluation of the new proposed mechanism Swift Gap is provided. From the evaluation, Swift Gap outperforms BGT, Conservative Backfilling (CONS) and Extensible Argonne Scheduling System (EASY).

Keywords: Swift Gap, Scheduling, Optimization, Completion Time

Introduction

In the computational grid, scheduling can be static or dynamic. The scheduling in the static mode is simple and straightforward; all the jobs arrive at a certain known of time to the system. The resources are always available for the whole scheduling process. Whereas in the dynamic mode, the jobs arrive to the system in different lengths of time, the scheduler has no idea about the arrival time until job reaching to the system. Moreover, the availability of the resources is not guaranteed. Most frequently, the candidate resources may sign out/in to the system for various reasons (such as resource failure).

The static mode is predictable for the scheduler and easy to implement. A traditional scheduling algorithm (such as; FCFS) (Henderson, 1995) can perform well for a small grid computing system. On the other hand, if thousands of jobs (that reach in different times) are waiting to be allocated for many available resources, more advanced policy that can deal with these circumstances is highly required.

Priority-based algorithms have been applied widely for scheduling purpose (Dakkak *et al*., 2006). The main issue of priority-based algorithms is maintaining the balance of performance regarding different metrics. Whereas; these algorithms can perform very well for specific metrics, while on the other hand they perform poorly compared to other aspects. For instance: Shortest Job First (SJF) (Davis and Patterson, 1975) has low flow-time and good utilization of the resources, but it has high makespan. Longest Job First (LJF) (Abraham *et al*., 2000) and Minimum Time to Due Date (MTTD) (Rasooli *et al*., 2008) have low makespan; conversely LJF suffers from high flow-time, high tardiness and low utilization of the resources.

In order to avoid these defects, several studies have implemented optimization algorithms such as: Ant Colony (Dorigo and Gambardella, 1997), Random Search (Solis and Wets, 1981) and Tabu Search (Glover and Laguna, 2013) alongside with priority algorithms. The previous applied approaches achieve better results and strikes a balance among the objective functions related to the end user. In addition, backfilling techniques were used for further exploiting of the available resources. Earliest Gap and Conservative Backfilling are examples of Backfilling techniques. Unlike the earliest gap policy (Klusácek and Rudová, 2008) which has to work alongside with based-scheduling algorithms, best-gap is able to perform alone. While Best Gap inherits the ability to reschedule the new arrival jobs without building a new scheduler

from EG and thus, Best Gap saving computational time compared to the previous mechanisms.

Even though Best Gap has the ability to build an incremental scheduler based on the new and existing jobs in the queue, but still scheduling thousands of jobs for numerous number of machines (resources), is not a smooth process and it consumes significant time at the expense of the QoS. Thus; a Meta-Heuristic mechanism that has the ability to optimize the scheduling is required in an environment such as in the grid. This paper presents an efficient integration between Best Gap and Tabu Search. Moreover, a new decision rule based on the Completion Time Weight is included. The outcome mechanism is named Swift Gap. Swift Gap mechanism has the ability to reduce the scheduling process by minimizing the objective functions with respect to Slowdown, Tardiness, Waiting Time and Response Time. Thus, Swift Gap mechanism has a remarkable improvement of QoS.

The rest of the paper is organized as follows: Section 2 introduces some of the related works. Section 3 describes Swift Gap structure. Section 4 explains the applied approach and the simulation settings. Section 5demonstrates the evaluation process and result analysis. Finally, this paper is concluded in section 6.

Related Work

The Backfilling technique refers to approach which avoids the fragmentation that caused by the small gaps among the jobs in the queue (or scheduler). These small gaps affect the utilization of resources by increasing the idle CPU time. EASY (Lifka, 1995) is the first mechanism that was developed to tackle this issue. EASY used First Come First Serve (FCFS) as a scheduling mechanism, in order to perform the scheduling. The main idea is to move the small job that can fit in the gap to an existing gap without affecting the first job that that located in the top of the queue.

Some researchers have implemented other priority-rules mechanisms (such as; SJF) with Backfilling technique. In (Tsafrir *et al*., 2007), Shortest Job First with Backfilling (SJFBF) was introduced. The main idea is to use EASY scheme to guarantee that no backfilled job will affect the first job at the top of the queue, while SJF will be applied for the jobs that will be backfilled.

In (Klusácek and Rudová, 2008), Earliest Gap-Earliest Deadline First (EG-EDF) was introduced followed by Tabu Search algorithm for further optimization. This strategy has a better performance from the previous ones due to the applied evaluation steps. In the schedule, when the short job arrives and if a gap is detected, the job will be moved into that gap using EG. If no gap detected, EDF policy will be applied alone. When the dispatching rule has to be decided to which machine the job should be allocated to, a Tabu search is implemented to calculate the possible job movement. Before moving the job, an evaluation based on the makespan and number of delayed jobs is applied. This evaluation helps to reduce the number of the moves. If the proposed move was better than the current one, the job will be moved, otherwise it will be maintained in the current position.

Other studies have implemented optimization search with/without priority search algorithm. (Somasundaram and Radhakrishnan, 2009) has implemented SJF with weighted random search. This mechanism achieved low residing time, but it suffers from high makespan. Integration between Ant Colony and Max-Min system in order to reduce the total computational time was proposed by (Nasira and Ku-Mahamudb, 2009).

Swift Gap Description

Swift Gap structure is based on CONS (Alem and Feitelson, 2001). In EASY, the backfilling is aggressive (i.e., EASY will check only if the backfilling will delay the first job in the queue). This could lead to long waiting time for the other jobs that are waiting in the queue. In conservative approach, the small jobs will be backfilled without causing any delay for the rest of jobs ahead in the schedule. This is achieved due to the runtime estimates. In this approach, the runtime estimates role (which is provided by the user) is very crucial to predict when each job will start and finish. Thus, the system will have the ability to expect the running time for the scheduled jobs and subsequently, the backfilling will be executed wisely. Whereas, EASY utilizes the runtime estimates. However, due to the aggressive approach to utilize the machine(s) more, EASY performs the backfilling considering only the delay for the first job in the queue.

The structure of Swift Gap consists of two kind of data. The first one includes the list of queued jobs in the system and the expected time to start. While the other structure maintains information about the expected resources usage in the future. In order to allocate the newly arriving jobs, an anchor point has to be detected. The anchor point is the point where the resources are available to tackle the backfilled jobs from start to finish without affecting any jobs in the schedule ahead. This information keeps updating itself every while to include/exclude the resources in/out the next backfilling cycle.

In reality, the runtime estimates provided by the users are so far from being accurate. Short runtime estimation will terminate the jobs, whereas long runtime estimation

will cause long waiting time for the jobs. To solve this problem, the runtime estimates will be exceeded to the maximum limit and then, the scheduler will compress itself to adapt the current situation. Moreover, the compression function helps to keep the original scheduler situation after backfilling decision is taken. This helps if a sudden termination of a job happened. In such a case, the job that ahead of the future backfilled job may take its place and that will lead to make the backfilled job waits for longer time in order to be backfilled again. Thus, the compression function exempts the scheduler from doing all backfilling calculations again by maintaining the scheduler situation as it is before the sudden termination of the running job happens. The only change occurs is the starting time and finishing time for the jobs. Therefore, in backfilling, all the jobs will start and finish earlier. Figure 1 illustrates the compressing function.

Swift Gap Approach

Our new proposed algorithm Swift Gap (Dakkak *et al*., 2016), adapting the decision rules form Best Gap policy, In addition, Swift Gap includes a new decision rule regarding the job movement in the schedule based on the completion time *(compl time)* (Gomoluch and Schroeder, 2003). The new decision rule *(compl time)* has shown a significant improvement in most of the QoS criteria for the end user. This paper will focus on four metrics, which are slowdown, tardiness, waiting time and response time.

A. Hybridization

There are several types of hybridization, which are: The high level and the low level. In the high level, hybridized algorithms are loosely coupled, whereas at the low level, the structures of algorithms are strongly coupled. Loosely coupled means when the first algorithm execution is over based on certain condition(s), the second algorithm starts in order to optimize the solution obtained from the first algorithm (Xhafa *et al*., 2011). In this study, the hybridization is based on loosely coupled concept. That means; both hybridized algorithms have their independent flow and commands i.e., no interaction between the structures of algorithms.

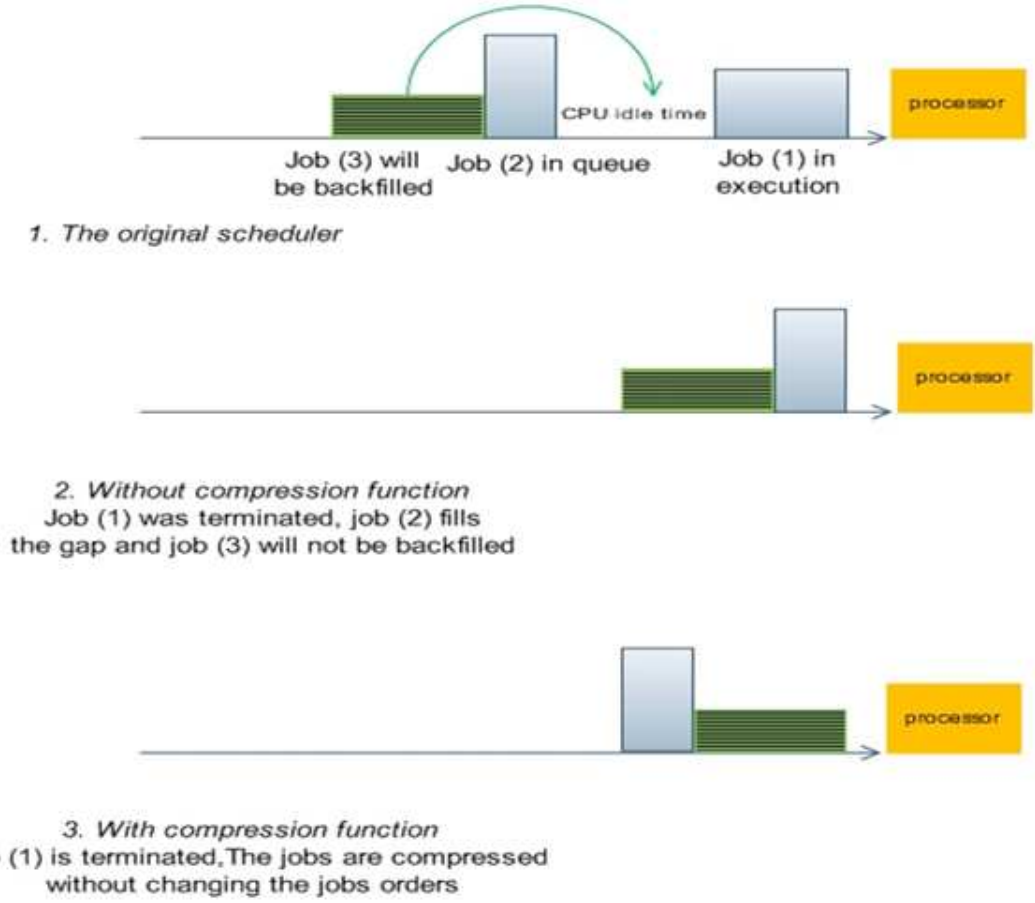

Fig. 1. The compressing function

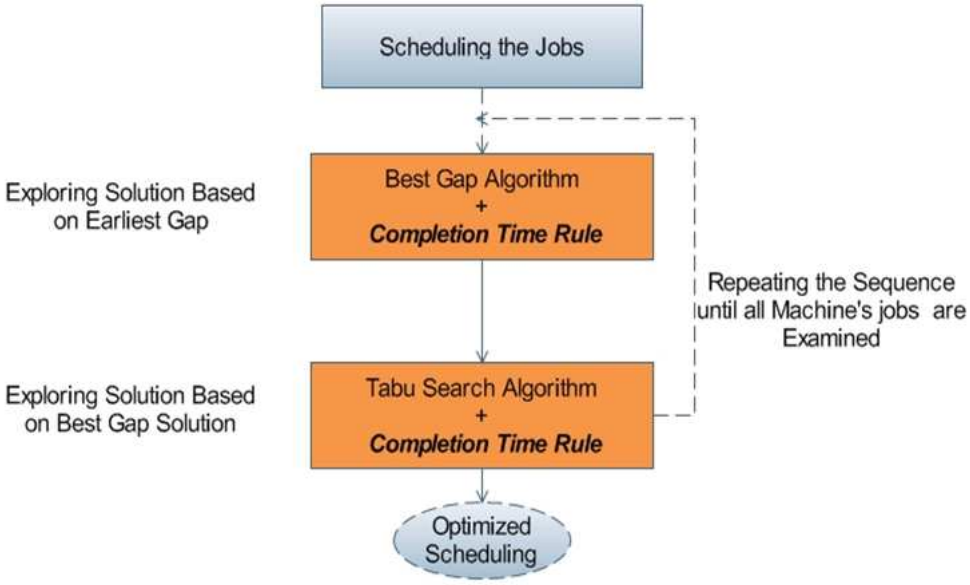

Fig. 2. Hybridization scheme for swift gap

Swift Gap has two stopping conditions. The first stopping condition is related to the creating the initial scheduler. When all new arrival jobs are tested in order to find the best gap, the initial scheduler is done and the optimizing stage starts. The second one in the optimization stage, where the mechanism will stop once required iterations' number is achieved or the time of is over. Figure 2 shows the hybridization scheme for Swift Gap with completion time rule included.

B. Swift Gap Working Steps

The scheduling in Swift Gap starts based on the gaps that are available in the scheduler, sensing of the gap(s) existence is based on the fragmentation measurement among the jobs in the schedule. When the gap is detected, the size of the job will be measured to determine if the job can be moved to the detected gap or not. If the size of the job is smaller or equal to the detected gap, the job will be eligible for the evaluation stage, which will be described later on. For further enhancement, the initial scheduling has to be improved through optimization mechanism. This could be done by moving the suitable job among the clusters rather than moving it within the computing cluster only. The optimized solution will subject to the same evaluation stage that applied in initial solution stage. Moreover, the iteration ability that optimization mechanism has; will enable Swift Gap to find the most optimized solution for the scheduling problem.

When Swift Gap has to take a decision for moving the job or keeping the job where it is (for evaluation stage), a one out of two options that Swift Gap has to take using the weight_function. This selection is based on the preferable value of the calculated metric (compl time) for decision (*A*) or (*B*). The resolution is taken upon this formula: ((Ametric-Bmetric)/Ametric). If the result of this formula is > 0.0, then Bmetric will be selected since minimizing these metrics is the better option. In other words, if the weight_function is greater than zero, (*B*) scheduling decision will be selected. Otherwise, if the previous formula is less than zero, the scheduler will schedule the job based on (*A*). A division by zero is prevented in the code. This evaluation of the weight_function for job moving is applied to detect which position will lead to a better performance metrics.

In Swift Gap, (*line 1-2*): The resources is created in the system. (*Line: 3-5*): The algorithm searching for a gap in the schedule to move the job into. If there is no gap, the algorithm will keep searching until finding a gap. The job will be moved based on completion time rule (*line: 6-7*). The evaluation of the move will be tested using the weight_function. Then, the executing of initial scheduling is over. (*line: 7-12*). The optimization search algorithm starts performing by adding the jobs in the schedule to its list in order to optimize them later (*line: 13-14*). The optimization of the jobs is conducted by manipulating the job's position among the clusters (*line: 15*). The movement of the job will be evaluated with respect to completion time weight. If the weight_function >0.0, then the current move (*B*) will be selected. Otherwise, the move will be rejected and previous move (*A*) will be selected (*line: 16-21*). Finally, the optimized order for jobs in the algorithm list will be sent to the allocated machine and the event will end (*line: 22*).

The weight_function is taking the scheduling decision depending on completion time (*line: 1*). After the calculation of the previous and current metrics is over, a discrimination between two scheduling decisions (*A* or *B*) will be executed. Decision (*A*) represents the previous move, whereas (*B*) decision represents the current move.

By reason of minimizing the metrics time in the Grid is highly desirable, the formula ((Ametric-Bmetric)/Ametric) determines which move is better (*A*) or (*B*). If ((Ametric- Bmetric)/Ametric)for the sum of all jobs is greater than zero, which means (*A*) value is greater than (*B*) value (*line: 2-6*). Therefore, the current move which is presented by (*B*) will be applied (*line: 7*), otherwise the previous move (presented by A) will take a place instead of (B) (line: 8-9). The event will end in (*line: 10*). Figure 3 presents the algorithm for Swift Gap and the weight_function.

Simulation and Discussion

The experiment is conducted using Alea Simulator (Klusáček and Rudová, 2010). Alea Simulator has a positive feedback from many researchers (Dakkak *et al*., 2015). The considered resource in our simulation is CPU. Two datasets are included in this experiment (Zewura and Wagap). The results reflect the real values for both dataset based on the jobs number. The number of the jobs is 3000, 5000, 7000, 9000, 10000, 15000 and 17500. The experiment is conducted using Intel I7-4770 CPU with 8 GB of RAM; the operating system is Windows 7.

Every simulation was repeated for 20 times. Figure 4-7 present the simulation results for slowdown, tardiness, waiting time and response time respectively for both datasets. Slowdown is a fraction (no unit), whereas tardiness, waiting time and response time are measured in seconds in the conducted simulation.

To evaluate the quality of the schedule offered by Swift Gap, different four criteria are used. The slowdown means how many times the job was delayed (ratio). The tardiness refers to the delay for the job related to certain due date. The waiting time refers to waiting time for the job has to wait in the schedule before it started to be processed. Response time is the running time for the job included to the waiting time for that job. Table 1 includes the parameters and configurations for the conducted simulation.

Table 1. Experiment parameters

Simulation settings	Info and values
Simulator	Alea 3.1
Number of Jobs	[3000-17500]
Simulation Time	3000
Simulated Resources	CPU
Runtime Estimates	True
Data-Set	Zewura, Wagap
OS	Windows 7

In Fig. 4, the completion time rule will make Swift Gap with less slowdown than the other mechanisms. Since (Gomoluch and Schroeder, 2003):

$$sld = \sum_{j=1}^{j} \frac{Completion\ Time - Submission\ Time}{Completion\ Time - Start\ Time} \quad (1)$$

Reducing the completion time, will also result in reducing the start time. Since start time appears in the denominator, thus, the total value of slowdown will certainly be minimized.

In Fig. 5, the tardiness will be decreased if the completion time is minimized as well, since:

$$Tardiness = \sum_{j=1}^{j} \max\left(0, Completion\ Time - Due\ Date\ Time\right) \quad (2)$$

While, due date time is fixed parameter in the workload, obviously decreasing the completion time will lead to minimize the total tardiness also. In Fig. 6, the waiting time in Swift Gap is less than BGT, CONS and EASY. The waiting time is the difference between the submission time and the start time. The start time is the time when the job is starting to be scheduled, whereas the submission time is the time when the job is submitted to the system. The reason behind this improvement, that the start time in Swift Gap is much less. Since (Gomoluch and Schroeder, 2003):

$$Tw = Start\ time - Submission\ Time \quad (3)$$

Reducing the completion time, will automatically lead in reducing the start time. Finally, in Fig. 7, it can be observed that the response time in Swift Gap is less compared to the other simulated algorithms. Since the response time is the sum of running time and waiting time:

$$Resp\ Time = Tr + Tw \quad (4)$$

The running time (*Tr*) for each job is a fixed value, while the waiting time (*Tw*) is already decreased as mentioned above, consequently, this will minimize the value of expected response time.

In the previous figures, it can be observed the variation in the interactions among the curve's patterns that produced by simulating the algorithms for zewura and wagap workloads. The reason behind this, that zewura workload has fewer resources than wagap workload. The presented completion time rule in Swift Gap has better effect when the number of the jobs is getting increases in wagap workload. This goes back to the huge resource number. This means that the completion time influence becomes less when the number of resources is huge for a limited number of the

jobs. Table 2 and 3; show the mean of the simulated objective functions for zewura and wagap workloads.

SG

```
1.    Add resources(r)to resource-in system scheduler(Rn);
2.    finish;
3.    check for the suitable gaps;
4.    if No gaps;
5.          Continue;
6.    if gap is detected;
7.          apply gap filling policy upon completion time;
8.    evaluate the move using weight_function;
9.    if weight_function > 0.0;
10.         select Scheduler B;
11.   else;
12.         Select Scheduler A;
13.   Start optimizing the initial solution
14.   Selecet non-optimized job and add it to the job list;
15.   move job;
16.   apply the optimizing upon completion time;
17.   evaluate the move using weight_function;
18.   if weight_function > 0.0;
19.         select Scheduler B;
20.   else;
21.         Select Scheduler A;
22.   end;
```

(a)

The Weight-function

```
1.    Decision is taken upon (completion time);
2.    calculating the preferable metric values for A and B starts;
3.    calculate the completion time for A;
4.    calculate the completion time for B;
5.    weight_function = weight (completion time)
6.    if  Σ_{j=1}^{j=n} (Acompl − Bcompl)/Acompl > 0;
7.          Select Scheduler B;
8.    else;
9.          Select Scheduler A;
10.   end if;
```

(b)

Fig. 3. (a) Swift gap (b) Weight function

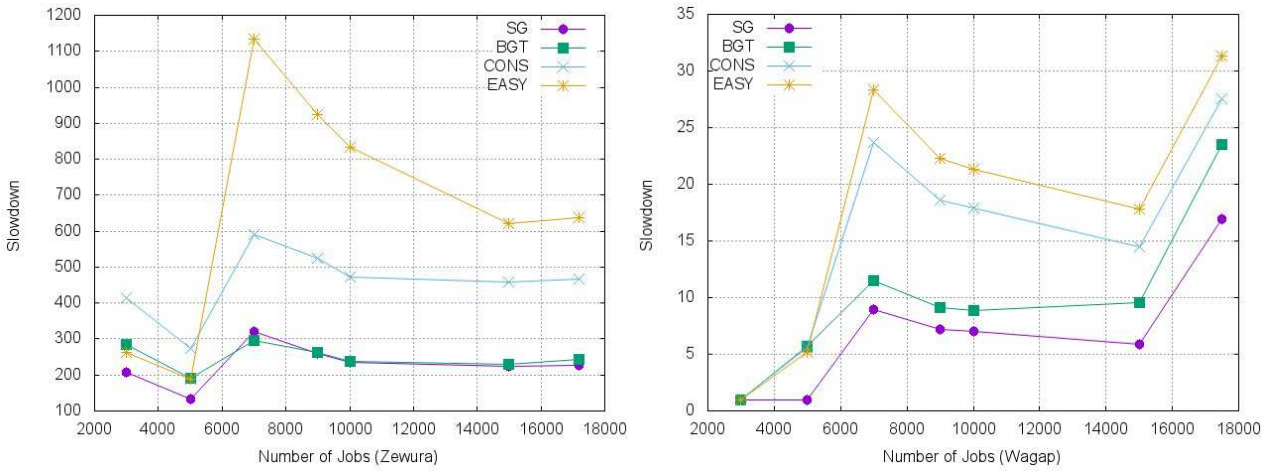

Fig. 4. Slowdown for Zewura and Wagap

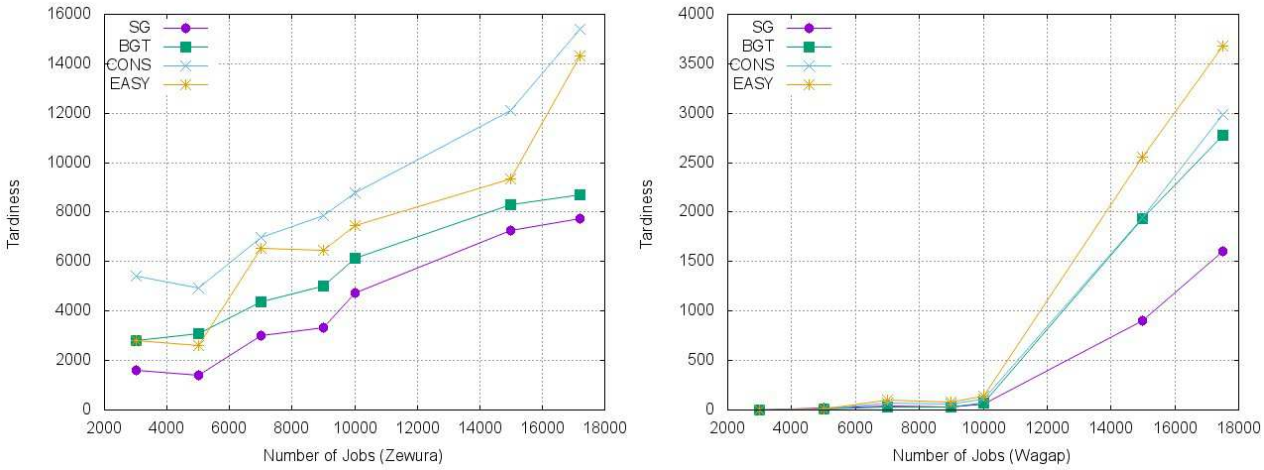

Fig. 5. Tardiness for Zewura and Wagap

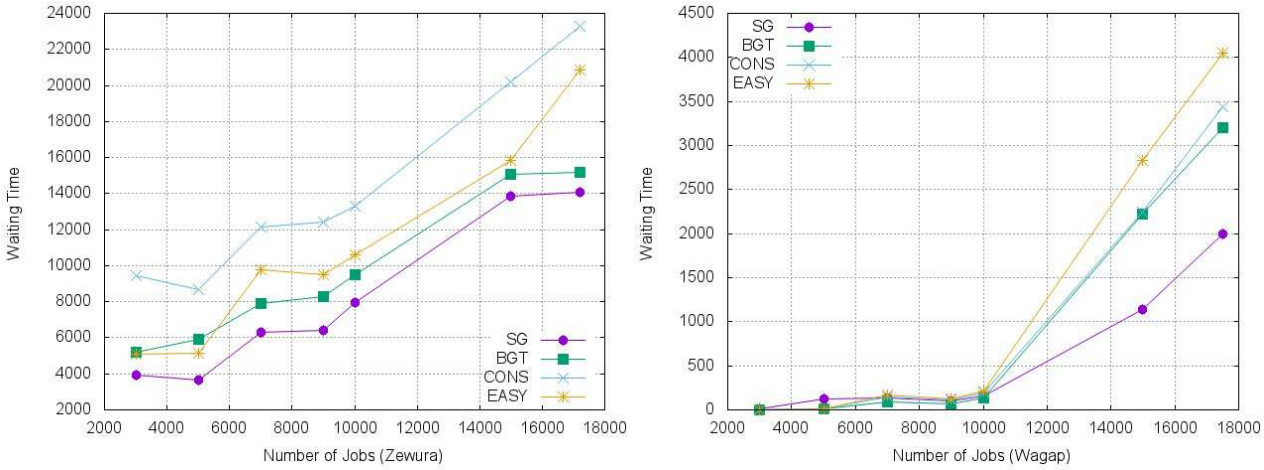

Fig. 6. Waiting Time for Zewura and Wagap

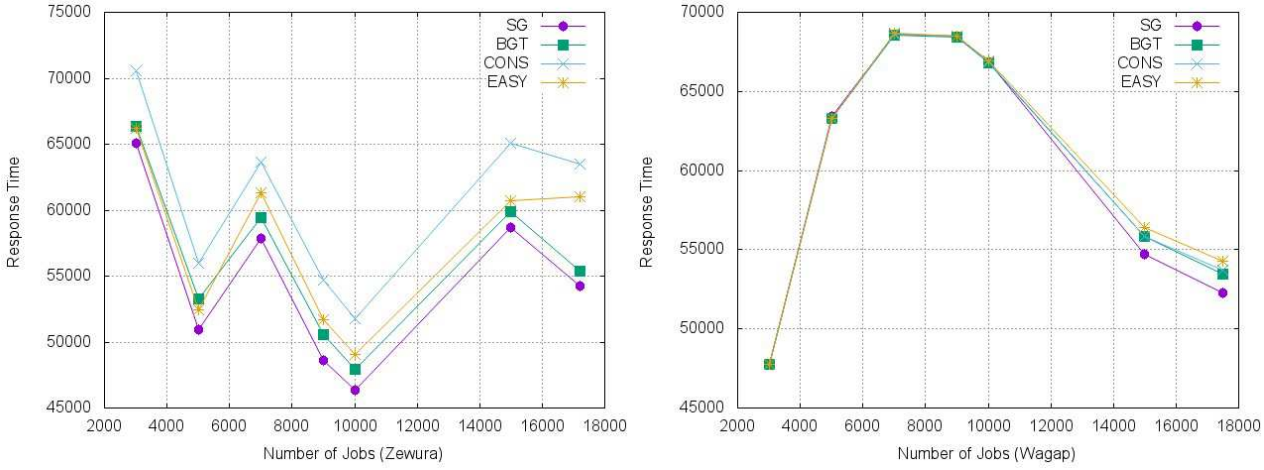

Fig. 7. Response Time for Zewura and Wagap

Table 2. Mean values of objective function (Zewura)

Metric	Mechanism	SG	BGT	CONS	EASY
Slowdown		229.6	249.0	457.6	657.7
Tardiness		4343.6	5492.2	8777.4	7066.7
Waiting Time		8021.7	9587.8	14218.7	10965.0
Response Time		54546.2	56112.2	60743.2	57489.5

Table 3. Mean values of objective function (Wagap)

Metric	Mechanism	SG	BGT	CONS	EASY
Slowdown		6.8	9.8	15.55	18.1
Tardiness		379.8	694.9	864.15	1092.4
Waiting Time		523.2	819.4	879.62	1081.3
Response Time		60300.4	60596.5	60656.70	60833.6

Table 4. Swift gap enhancement ratio

Metric	Mechanism	$BGT_{(zewura)}$	$BGT_{(wagap)}$	$CONS_{(zewura)}$	$CONS_{(wagap)}$	$EASY_{(zewura)}$	$EASY_{(wagap)}$
Slowdown		+8.123	+36.2224	+66.3679	+77.6786	+96.4962	+90.4438
Tardiness		+23.3552	+58.6401	+67.5818	+77.8684	+47.7301	+96.808
Waiting Time		+17.7861	+44.1166	+55.727	+50.8058	+31.0031	+69.5625
Response Time		+2.8304	+0.4899	+10.7503	+0.5892	+5.2541	+0.8804

From Table 2 and 3, it can be noticed the enhancement offered by Swift Gap, compared to our hybridization without including the completion time rule (BGT), CONS and EASY in Table 4. Obviously, Swift Gap outperform BGT, CONS and EASY in all objective functions for both workloads. Even though, the completion rule influence is not so obvious in wagap workload when the number or jobs is few, but later, when the number of the jobs is getting increased, the enhancement of completion time rule becomes very conspicuous and even much better than zewura. Whereas; in zewura workload, the role of the included completion time in Swift Gap can be observed once the simulation launches. This dissimilar behavior of Swift Gapand even for the other simulated algorithms between the different workloads can be justified based on the properties of the workload. As discussed earlier, the number of resources and the size of the jobs for each workload can be one of many reasons behind this dissimilar behavior.

Finally, it can be observed that all objective functions behaviors in the simulated dynamic case are not stable (nonlinear curve). That is due to the different arrival times for the jobs and different waiting times as well. Thus, the variation in the curves (oscillatory behavior) is observed regardless the number of jobs.

Conclusion

In this study, Swift Gap mechanism is presented. Swift Gap exploits the features of Best Gap and Tabu Search. In addition, a completion time rule has included into the mechanism. The completion time rule has proven its efficiency by minimizing the objective functions. The simulation is conducted using Alea Simulator. The experiments were carried out for two datasets for a huge jobs' number.

The simulations have shown that Swift Gap has minimized the slowdown, tardiness, waiting time and response time thanks to the completion time rule. Moreover and from the presented evaluation, Swift Gap outperforms Best Gap with Tabu Search (BGT), CONS and EASY in all simulated performance metrics.

Acknowledgement

Zewura and Wagap workloads log were graciously provided by the Czech National Grid Infrastructure MetaCentrum.

Funding Information

This work is funded by UNIVERSITI UTARA MALAYSIA (UUM) S/O Code: 15861.

Author's Contributions

As the first author of this paper, I would like to confirm that this study is a part of my PhD work; whereby I have been closely supervised by Dr. Shahrudin Awang Nor and Dr. Ahmad Suki Che Mohamed Arif.

Ethics

This article is original and contains unpublished material. The corresponding author confirms that all of the other authors have read and approved the manuscript and no ethical issues involved the other authors have read and approved the manuscript and no ethical issues involved.

References

Abraham, A., R. Buyya and B. Nath, 2000. Nature's heuristics for scheduling jobs on computational grids. Proceedings of the 8th IEEE International Conference on Advanced Computing and Communications, (ACC' 00).

Alem, A.W.M. and D.G. Feitelson, 2001. Utilization, predictability, workloads and user runtime estimates in scheduling the IBM SP2 with backfilling. IEEE Trans. Parallel Distributed Syst., 12: 529-543. DOI: 10.1109/71.932708

Dakkak, O., S. Arif and S.A. Nor, 2006. Resource allocation mechanisms in computational grid: A survey.

Dakkak, O., S. Arif and S.A. Nor, 2015. A critical analysis of simulators in grid.

Dakkak, O., S.A. Nor and S. Arif, 2016. Proposed algorithm for scheduling in computational grid using backfilling and optimization techniques. J. Telecommun. Electr. Comput. Eng., 8: 133-138.

Davis, E.W. and J.H. Patterson, 1975. A comparison of heuristic and optimum solutions in resource-constrained project scheduling. Manage. Sci., 21: 944-955. DOI: 10.1287/mnsc.21.8.944

Dorigo, M. and L.M. Gambardella, 1997. Ant colony system: A cooperative learning approach to the traveling salesman problem. IEEE Trans. Evolut. Comput., 1: 53-66. DOI: 10.1109/4235.585892

Glover, F. and M. Laguna, 2013. Tabu Search. Springer.

Gomoluch, J. and M. Schroeder, 2003. Market-based resource allocation for grid computing: A model and simulation. Proceedings of the Middleware Workshops, (MW' 03).

Henderson, R.L., 1995. Job scheduling under the portable batch system. Proceedings of the Job Scheduling Strategies for Parallel Processing, (SPP' 95), Springer-Verlag, pp: 279-294.

Klusácek, D. and H. Rudová, 2008. Improving QoS in computational grids through schedule-based approach. Proceedings of the Scheduling and Planning Applications Workshop at the Eighteenth International Conference on Automated Planning and Scheduling, (APS' 08), Sydney, Australia.

Klusáček, D. and H. Rudová, 2010. Alea 2: Job scheduling simulator. Proceedings of the 3rd International ICST Conference on Simulation Tools and Techniques, Mar. 15-19, ICST, Brussels, Belgium. DOI: 10.4108/ICST.SIMUTOOLS2010.8722

Lifka, D.A., 1995. The ANL/IBM SP scheduling system. Proceedings of the Workshop on Job Scheduling Strategies for Parallel Processing, (SPP' 95), Springer-Verlag, London, pp: 295-303.

Nasira, H.J.A. and K.R. Ku-Mahamudb, 2009. Hybrid ant colony optimization for grid computing.

Rasooli, A., M. Mirza-Aghatabar and S. Khorsandi, 2008. Introduction of novel rule based algorithms for scheduling in grid computing systems. Proceedings of the 2nd Asia International Conference on Modeling and Simulation, May 13-15, IEEE Xplore Press, pp: 138-143. DOI: 10.1109/AMS.2008.83

Solis, F.J. and R.J.B. Wets, 1981. Minimization by random search techniques. Math. Operat. Res., 6: 19-30. DOI: 10.1287/moor.6.1.19

Somasundaram, K. and S. Radhakrishnan, 2009. Task resource allocation in grid using swift scheduler. Int. J. Comput. Commun. Control, 42: 158-166. DOI: 10.15837/ijccc.2009.2.2423

Tsafrir, D., Y. Etsion and D.G. Feitelson, 2007. Backfilling using system-generated predictions rather than user runtime estimates. IEEE Trans. Parallel Distributed Syst., 18: 789-803. DOI: 10.1109/TPDS.2007.70606

Xhafa, F., J. Kołodziej, L. Barolli and A. Fundo, 2011. A GA+TS hybrid algorithm for independent batch scheduling in computational grids. Proceedings of the 4th International Conference on Network-Based Information Systems, Sept. 7-9, IEEE Xplore Press, pp: 229-235. DOI: 10.1109/NBiS.2011.41