COMPUTATIONAL SCIENCE AND TECHNIQUES
Volume 8, 2021, 631-638

Publisher: Klaipėda University
https://e-journals.ku.lt/journal/csat
Online ISSN: 2029-9966
DOI: 10.15181/csat.v8.2103

TASK STALLING FOR A BATCH OF TASK MAKESPAN MINIMISATION IN HETEROGENEOUS MULTIGRID COMPUTING

Albertas Jurgelevičius[1], Leonidas Sakalauskas[2], Virginijus Marcinkevičius[3]
[1]albertas.jurgelevicius@mif.vu.lt
Institute of Data Science and Digital Technologies, Vilnius University, Lithuania
[2]Vilnius Gediminas Technical University, Lithuania,
Institute of Data Science and Digital Technologies, Vilnius University, Lithuania
[3]Institute of Data Science and Digital Technologies, Vilnius University, Lithuania

Abstract. This paper presents a new algorithm for a batch of task makespan minimisation in heterogeneous multigrid computing. Heterogeneous grids are known to cause straggling task problem that increases task execution makespan. Existing task distribution algorithms solve this problem by using information about the compute node capacities or task sizes. However, such information may not always be available. Task stalling solves both problems. However, this method is described for queuing systems consisting of only two heterogeneous servers or grids. Our proposed algorithm is based on an improved task stalling method, allowing it to distribute tasks in systems consisting of two or more grids. Experiment results show reduced task execution makespan by up to 19,92% compared to FIFO. This allows us to conclude that the new algorithm is suitable for a batch of task makespan minimisation in heterogeneous multigrid computing.

Keywords: heterogeneous multigrid computing, a batch of task makespan minimisation, scheduling algorithm

1. Introduction

The demand for computing resources in businesses companies is significantly increasing since the introduction of the Internet of Things and smart technologies. Computing in modern businesses is usually limited to the resources of available servers or cloud computing services. However, research shows that a significant part of their internal computing resource capacity is only lightly loaded. Since there are no distributed computing platforms designed for companies to use all available computing resources, this causes the inefficient use of their information technologies.

In this paper, we present a new algorithm for distributing tasks in heterogeneous hybrid distributed computing platforms. The proposed algorithm minimises task execution makespan by using the task stalling method. This method performs straggling task mitigation. The straggling task problem is one of the main reasons preventing business companies from using all internal computing resources [1]. The task stalling method does not use or require information about compute node capacities or task sizes. However, this method is described for queuing systems consisting of only two heterogeneous servers or grids. The proposed

algorithm is based on a modified version of the existing task stalling method to support systems consisting of two or more grids.

This paper is structured as follows. Section 2 presents related work and outlines the novelty of our proposed algorithm. Section 3 explains the task stalling method and presents the proposed algorithm. Section 4 contains experiments. Finally, experiment results are concluded in Section 5.

2. Related Work

Hybrid distributed computing solutions combine public and private computing grids to improve computation efficiency and reliability [2]. Tasks between private and public grids can be distributed using various existing task distribution algorithms (see Table 1). However, our review showed that all existing task distribution methods use information about the batch of task size, estimated task execution times, computational resource capacity. This data is used to schedule tasks. However, in a heterogeneous environment, these parameters are constantly changing or cannot be estimated. This means that in such a case, existing solutions cannot be used.

Table 1. List of related works.

Algorithm	Year	Method	Optimisation criteria
Topcuoglu, H. et al. [3]	2002	The highest priority tasks are assigned to recourses that can soonest deliver the results.	Makespan minimisation.
Bittencourt, L. F. et al. [4]	2011	Improved HEFT [3] algorithm.	Makespan minimisation, task completion within the deadline.
Bittencourt, L. F. et al. [5]	2012	The optimisation is done using linear programming.	Makespan minimisation, task completion within the deadline.
Vecchiola, C., et al. [6][7]	2012	Created an algorithm that assigns more recourses to slow running tasks.	Task completion within the deadline.
Van den Bossche, R. et al. [8]	2013	Created rules defining which task distribution algorithm to use.	Makespan minimisation, task completion within the deadline.
Duan, R. et al. [9]	2014	The optimisation is done using game theory.	Makespan and cost minimisation.
Wang, B. et al. [10]	2016	Created a new algorithm that distributes tasks based on calculation cost to efficiency ratio.	Resource utilisation.
Zhang, Y. et al. [11]	2017	Improved PSO algorithm.	Task completion within the deadline, cost minimisation.
Abdi, S. et al. [12]	2017	The optimisation is done using linear programming.	Makespan minimisation, task completion within the deadline.
Zhang, Y. et al. [13]	2019	The optimisation is done using "Firefly" algorithm.	Makespan and cost minimisation.
Zhang, Y. et al. [14]	2019	Created a new algorithm for task re-distribution.	Makespan and cost minimisation.
Stavrinides, G. L. et al. [2]	2021	Improved Min-Min and Min-Max algorithms [15][16].	Task completion within the deadline, cost minimisation.

3. Task Stalling buffer

Task stalling buffer [17] reduces task execution makespan in queuing systems consisting of two heterogeneous servers (see Figure 1). It reduces slow server load by distributing a larger part of the incoming new tasks to the fast server. If the fast server is not available, then tasks are added to the task stalling buffer. If the task stalling buffer is full, then tasks are distributed to the slow channel. For example, such a queuing system can be used to distribute tasks between private and public grids (company servers and desktop computers). Task stalling buffer length K can be calculated using equations from 1 to 5 [17].

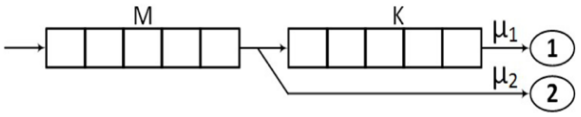

Figure 1. Queuing system with task stalling buffer [17], where 1 is the fast server and 2 is the slow server. Variable definitions are presented in Table 2.

Table 2. Variable descriptions.

Variable	Description
M	Task waiting buffer length.
K	Task stalling buffer length.
μ_1	Fast channel efficiency.
μ_2	Slow channel efficiency.
r	Efficiency ratio between fast and slow channels.
q	Task execution efficiency coefficient.
c	The number of completed tasks.
t	Sum of task execution times.
m	The number of nodes in the fast channel.
a_1	The number of completed tasks by the fast channel.
b_1	Time required to complete tasks by the fast channel.
a_2	The number of completed tasks by the slow channel.
b_2	Time required to complete tasks by the slow channel.

$$K \approx r \cdot (1 - q)\,, \tag{1}$$

$$r = \frac{\mu_1}{\mu_2}, \tag{2}$$

$$q = \frac{c}{t \cdot m \cdot \mu_1}\,, \tag{3}$$

$$\mu_1 = \frac{a_1}{b_1}\,, \tag{4}$$

$$\mu_2 = \frac{a_2}{b_2}\,. \tag{5}$$

Algorithm 1: task distribution

Input: A bag-of-task job $J = \{t_1, t_2, \dots, t_m\}$ *of* m *tasks*

Output: none

1. $clusters \leftarrow$ *order clusters by each cluster's efficiency* μ_i $(1 \leq i \leq n)$
2. $lengths \leftarrow$ *calculate buffer length* k_i *for each buffer* b_i $(1 \leq i < n)$
3.
4. ***for each*** *task* $t_j \in J$ ***do***
5. ***repeat***
6. ***for each*** *cluster* $c_i \in clusters$ ***do***
7. ***if*** *cluster* n_i *accepts new tasks* ***then***
8. *schedule task* t_j *to cluster* n_i
9. ***else if*** $i < n$ ***and*** *number of tasks in buffer* $b_i < k_i$ ***then***
10. *add task* t_j *to buffer* b_i
11. ***end if***
12. ***end for***
13. ***until*** *task* t_j *is distributed*
14. ***end for***

Similarly, task stalling buffer can be used in systems with more than two channels (see Figure 2). For example, such a queuing system can be used to distribute tasks between multiple private and public grids (company servers, cloud services and desktop computers). Task stalling reduces task execution makespan the most when the performance difference between two grids is the largest [17]. In a system with n $(n > 1)$ grids, tasks must be distributed between the slowest grid (the slow channel) and the rest of $n - 1$ grids (the fast channel). This means that equation no. 1 can still be used for calculating the buffer length K (see Algorithm 1). This process can be illustrated using Figure 2. Here, "Grid no. 1" (the slow channel) performs computations the slowest of all grids, whereas "Grid no. 3" performs computations the fastest. "Grid no. 2" performs computations slower than "Grid no. 3" but faster than "Grid no. 1". If "Grid no. 3" is busy, new tasks are distributed to the task stalling buffer $K2$ (the buffer length is calculated using equation no. 1). If the task stalling buffer $K2$ is full, tasks are distributed to "Grid no. 2". If "Grid no. 2" is busy, new tasks are distributed to the task stalling buffer $K1$ (the buffer length is calculated using equation no. 1). If the task stalling buffer $K1$ is full, tasks are distributed to "Grid no. 1". If "Grid no. 1" is busy, new tasks are stored in the waiting buffer until tasks can be distributed either to the task stalling buffer $K1$ or "Grid no. 1".

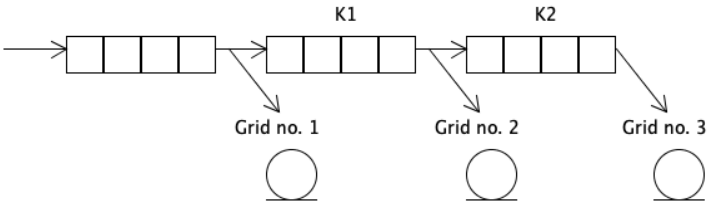

Figure 2. Task stalling buffer in a queuing system consisting of 3 grids.

4. Experiment

The purpose of the experiment is to check the hypothesis that the proposed task distribution algorithm reduces the task execution makespan more than the standard FIFO (first in, first out), also known as FCFS (first come, first served) algorithm. Experiment results are tested against FIFO, since there are no other suitable task distribution methods (as discussed in Section 2). Tests are executed using our created virtual environment for simulating grids (developed using PHP programming language). Furthermore, the following algorithm modifications are tested:

a) Static length task stalling buffer. Buffer length is calculated only once.
b) Dynamic length task stalling buffer. Buffer length is re-calculated after each new task is submitted to the system.

Pre-generated scenarios (see Table 3) are used in experiments to compare task execution makespan between different algorithms. Scenarios are based on the following two parameters: task size and task start time. Both parameters are measured in iterations (pseudo-operations). All scenarios are configured to induce a reasonable system load. In this case, the task stalling buffer should not constantly be empty or full. Otherwise, one of the following scenarios would occur:

a) All of the tasks would be redirected to the fast channel, and the system would be underutilised.
b) All of the tasks would be redirected to the slow channel, and the task stalling method would behave exactly like the FIFO algorithm.

Various cluster configurations (see Table 4) were used to test how the algorithms would perform in various environments. Experiment no. 1 and 2 executes using all possible batch of task sizes, ranging from 10 to 100 tasks. Experiment no. 3 executes using all possible batch of task sizes, ranging from 40 to 400 tasks. After each experiment, results from each scenario are aggregated. All agents in Cluster D have 1000 iteration task start penalty (simulating virtual machine load time).

Experiment results show (see Figures 3 and 4) that the dynamic size task stalling buffer reduces task execution makespan in all scenarios compared to FIFO. Static size task stalling buffer in *TS_STS* scenario increased task execution makespan by 3.05% compared to FIFO, because slow channel recourses were underutilised due to the static buffer size.

Table 3. Annotations used to define the experiment scenarios.

Notation	Description
TS	Static size tasks. All tasks are of the same size (200 iterations).
TD	Dynamic size tasks. Generated task sizes are distributed using Poisson distribution ($\boldsymbol{\lambda = 200}$).
STS	Static intensity stream. The delay between each task start time is 8 iterations.
DTS	Dynamic intensity stream. The delay between each task start time is generated using Poisson distribution ($\boldsymbol{\lambda = 8}$).
TS_STS	Static size tasks (***TS***), static intensity stream (***STS***). Queuing system is sent uniform tasks in equal intervals.
TS_DTS	Static size tasks (***TS***), dynamic intensity stream (***DTS***). Queuing system is sent uniform tasks in varying intervals.
TD_STS	Dynamic size tasks (***TD***), static intensity stream (***STS***). Queuing system is sent varying size tasks in equal intervals.
TD_DTS	Dynamic size tasks (***TD***), dynamic intensity stream (***DTS***). Queuing system is sent varying size tasks in varying intervals.

Table 4. Experiment cluster configurations.

Cluster	Experiment no. 1	Experiment no. 2	Experiment no. 3
A	2 agents.	2 agents.	5 agents.
B	2 agents, that are 3 times slower than agents in cluster A.	2 agents, that are 6 times slower than agents in cluster A.	5 agents, that are 3 times slower than agents in cluster A.
C	2 agents, that are 9 times slower than agents in cluster A.	2 agents, that are 6 times slower than agents in cluster A.	5 agents, that are 6 times slower than agents in cluster A.
D	2 agents, that are 27 times slower than agents in cluster A.	2 agents, that are 27 times slower than agents in cluster A.	20 agents, that are 27 times slower than agents in cluster A.

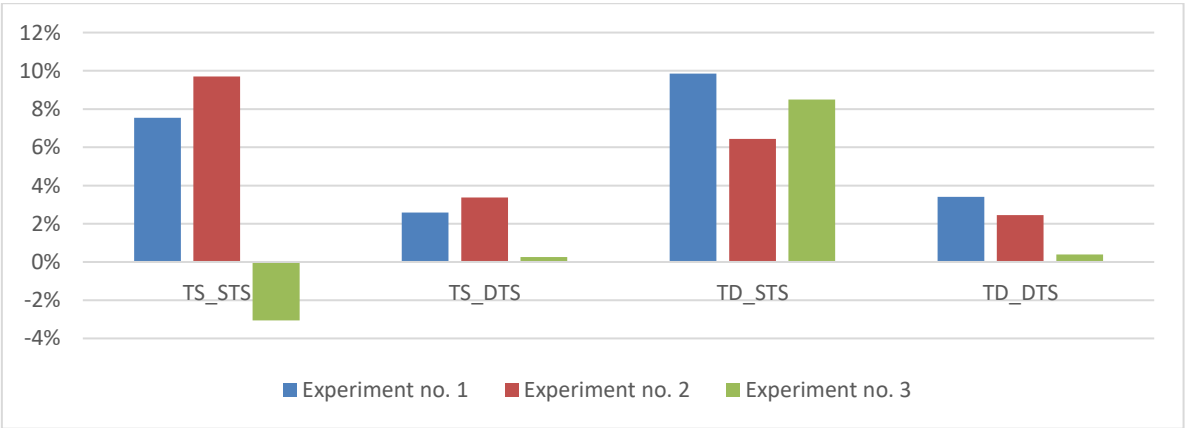

Figure 3 Task execution makespan reduction using static size task stalling buffer, compared to FIFO.

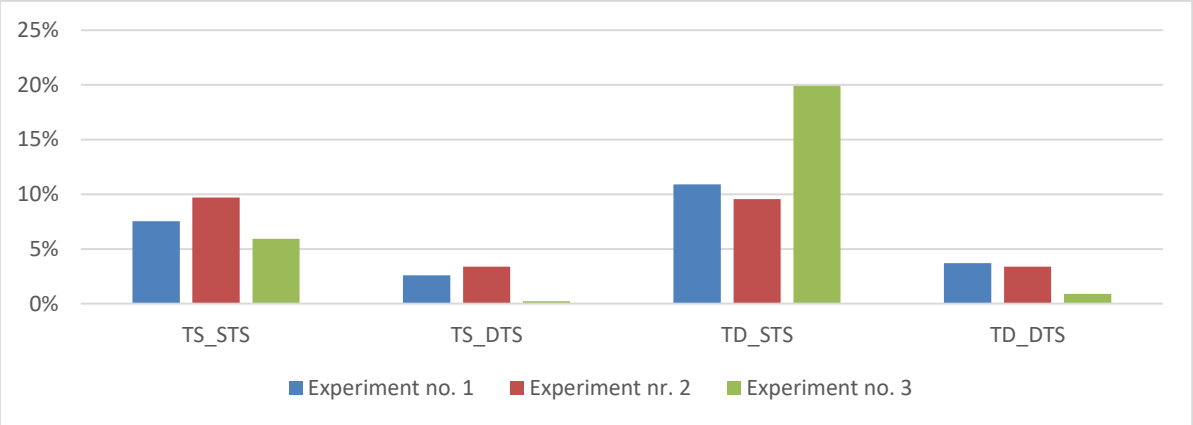

Figure 4. Task execution makespan reduction using dynamic size task stalling buffer, compared to FIFO.

5. Conclusions

Experiment results showed that the proposed algorithm with dynamic task stalling buffer reduces task execution makespan up to 19,92% compared to FIFO. The proposed algorithm works best when the incoming new task stream is static. Experiment results showed that the dynamic length task stalling buffer performs better than the static length task stalling buffer. This allows us to conclude that the new algorithm with dynamic length task stalling buffer should be used for a batch of task makespan minimisation in heterogeneous multigrid computing, where no information about the tasks or the compute nodes is available.

References

1. Gill, S. S., Ouyang, X., Garraghan, P., Tails in the cloud: a survey and taxonomy of straggler management within large-scale cloud data centres. The Journal of Supercomputing, 2020, vol. 76, 10050–10089. DOI: 10.1007/s11227-020-03241-x.
2. Stavrinides, G. L., Karatza, H. D., Dynamic scheduling of bags-of-tasks with sensitive input data and end-to-end deadlines in a hybrid cloud. Multimedia Tools and Applications, Springer, 2021, vol. 80, 16781–16803.
3. Topcuoglu, H., Hariri S., Wu M.-Y., Performance-effective and low-complexity task scheduling for heterogeneous computing. Transactions on Parallel and Distributed Systems, IEEE , 2002, 13(3), 260–274.
4. Bittencourt, L. F., Madeira, E. R. M., HCOC: A cost optimization algorithm for workflow scheduling in hybrid clouds. Journal of Internet Services and Applications, 2011, vol. 2, 207–227.
5. Bittencourt, L. F., Madeira, E. R. M., Fonseca, N., Scheduling in Hybrid Clouds. IEEE Communications Magazine, 2012, vol. 50.
6. Calheiros, R. N., Vecchiola, C., Karunamoorthy, D., Buyya, R., The Aneka platform and QoS-driven resource provisioning for elastic applications on hybrid Clouds. Future Generation Computer Systems, 2012, 28(6), 861-870, DOI: 10.1016/j.future.2011.07.005.
7. Vecchiola, C., Calheiros, R. N., Karunamoorthy, D., Buyya, R., Deadline-driven provisioning of resources for scientific applications in hybrid clouds with Aneka. Future

Generation Computer Systems, 2012, 28(1), 58-65, DOI: 10.1016/j.future.2011.05.008.

8. Van den Bossche, R., Vanmechelen, K., Broeckhove, J., Online cost-efficient scheduling of deadline-constrained workloads on hybrid clouds. Future Generation Computer Systems, 2013, 29(4), 973–985.
9. Duan, R., Prodan, R., Li, X., Multi-Objective Game Theoretic Schedulingof Bag-of-Tasks Workflows on Hybrid Clouds. Transactions on Cloud Computing, 2014 IEEE, vol. 2, 29-42, DOI: 10.1109/TCC.2014.2303077.
10. Wang, B., Song, Y., Sun, Y., Liu, J., Managing Deadline-Constrained Bag-of-Tasks Jobs on Hybrid Clouds. Society for Computer Simulation International, 2016, USA.
11. Zhang, Y., Sun, J., Novel efficient particle swarm optimization algorithms for solving QoS-demanded bag-of-tasks scheduling problems with profit maximization on hybrid clouds. Concurrency and Computation: Practice and Experience, 2017, vol. 29.
12. Abdi, S., PourKarimi, L., Ahmadi, M., Zargarid, F., Cost minimization for deadline-constrained bag-of-tasks applications in federated hybrid clouds. Future Generation Computer Systems, 2017, vol. 71, 113–128.
13. Zhang, Y., Zhou, J., Sun, L., Mao, J., Sun, J., A Novel Firefly Algorithm for Scheduling Bag-of-Tasks Applications Under Budget Constraints on Hybrid Clouds. IEEE Access, 2019, vol. 7, 151888-151901, DOI: 10.1109/ACCESS.2019.2948468.
14. Zhang, Y., Zhou, J., Sun, J., Scheduling bag-of-tasks applications on hybrid clouds under due date constraints. Journal of Systems Architecture, 2019, vol. 101, 101654, DOI: 10.1016/j.sysarc.2019.101654.
15. Bhatia, M., Task Scheduling in Grid Computing: A Review, Advances in Computational Sciences and Technology, ISSN 0973-6107, vol 10(6), 2017 pp. 1707-1714.
16. Dong, F., Akl, S.G., Scheduling algorithms for grid computing: state of the art and open problems. Technical Report No. 2006-504, School of Computing, Queen's University, Kingston, Ontario, January 2006.
17. Kaklauskas, L., Sakalauskas, L., Denisovas, V., Stalling for solving slow server problem. RAIRO - Operations Research, vol. 53 (4), July 2018, DOI: 10.1051/ro/2018056.