Analysis of Work Stealing Approaches for Load Balancing in Distributed Environment

TanuShree[1*], and Neelendra Badal[2]
[1]*Department of Computer Science, IFTM University Moradabad, India*
[2]*Computer Science Department,KNIT, Sultanpur, India*

***Abstract*:** *Spontaneously allocated duties and their execution are one of the biggest duties of the processor. But it can be observed some time that the overall load is not divided equally, therefore the potentiality of the overall execution system can be lower. Therefore, the work-stealing process is present where the load becomes balanced in each of the cores effectively. The basic topology of the whole work-stealing process is very simple. Cores from different regions of the process have access to steal work whenever it becomes necessary. In case of any necessity when the load becomes heterogenic, the cores steal the load and make it balanced effectively. Different procedures are present for example the organizational SSL, SLL, LLL, and LLS. Through these, the work-stealing procedure can be done according to the work level or the priority.*

***Keywords*:** *Load balancing, work stealing, distributed system.*

1. Introduction

Work stealing is a very popular load balancing technique in the world of dynamic parallelism. It is used to maintain a pool of workers, each of which maintains a double-ended queue or the deque of tasks. Generally, the theory of work-stealing methodology comes from the fact of the whole process where the local deque becomes empty. However, it may happen that the victim is also running out of work. Thereby, this becomes a more dramatic case, where the thief also runs out of the work or jobs. Therefore this situation is known as the failed steal attempt, as the thief is unable to receive any job by applying the process of work Stealing.

In this generation, supercomputers are coming with a lot more power delivery systems, including multicore processors, a wide mixture of the shared memory and distributed memory parallelism. There, a lot of work-stealing processes are present. The whole research is aimed to identify the proper aspects of the work scheduling model and comparing the possible work-stealing methods. Discussing the advantages and disadvantages and the application of the theories are having the highest priority in this whole research during the comparison.

The research is going to be done in the following way where the systematic design will follow the basic structure of details about the specific algorithms and comparison between those with the help of several aspects.

* Corresponding Author

2. Work stealing

Work Stealing is a process which was proposed in the market regarding the execution of the functional programs on the virtual tree of multithreaded processors. Almost in every case parallel computing provides better output and performance measurements. The pre-installation environment or the PE of the computing environment plays the main role in the work-stealing process. Different procedures are present to do the overall process [1]. The main algorithm works based on the following structure. Generally, PE searches for the existing work or task for execution. If the PE finds any task then it will acquire that and take that for the next level processing. At the same time if the PE does not find any type of tasks in the computing environment it throws a stealing signal to another appropriate PE. If the PE does have any type of remaining task then the blank PE will steal and execute that whereas, if the other PE also remains blank then the stealing signal reverts to a blank signal or 0. In this scenario, the PE fails to steal the task from another PE in the computing environment. The whole process is known as work stealing [2]. The case where the stealing process reverts the signal 0 or null is known as the failure of the stealing process. On the other hand, when the stealing process has more than one choice, the PE will select all the choices and send those to the thief directly from where the stealing attempt was initiated. The simplest version of the stealing process is the Random stealing process, where the targets for steal attempts are chosen in a randomized way. In this process, the stealing target faces a lot of attempts but the victim sends a single task as a response to the thief [3].

3. Different types of work stealing

Generally, in the world of work-stealing, there are mainly four types of processes present. SSL, SLL, LLL, LLS are the four parts of work-stealing. All the parts are described below.

3.1. SSL

The SSL stands for the small-small-large distributed system. When it is important to respond to a still attending from the same type of PE for the same type of cluster level, it is important to select the smallest task. In this SL system, the victim always wants to keep the thieves busy remotely for a longer period, than the pre-installation environment from the same cluster. It happens because it pushes the themes to request for work by having a longer duration with the previous one. The overall process is done through a very high latency network. Therefore the thief becomes forceful to send the Steel attempts signal in terms of continuing the overall process of parallel computing in the distributed system [4].

3.2. SLL

The SLL stands for a small large large system. Choosing the smallest task only for the same pre-installation environment level and selecting the biggest task for all other levels are considered in this type of distributed system. Therefore the overall system of choosing the performance level of the parallel computing is very much affected by this system in the distributed system. Generally, when the overall system is having a theft attempt to the main victim from the side of the thief regarding the work-stealing, the PE or the pre-installation environment does not care about the sources of the attempt. That means the overall functionalities of the SLL does not have a focus on the clusters. It can be the same cluster or a different one [5]. On the other hand, it specifies the largest tasks for the other PEs as well.

Therefore it can be understood that the effectiveness of the SLL is very much associated with the overloads recovery of every possible corner that a distributed system used to have. For example, let's assume that one corner of the system is heavily loaded, whereas the other corner of the system is lightly loaded. Therefore it becomes an unequal situation. The work-stealing is the process which is used to push a theft attempt and steal some of the work from the heavily loaded corner and distribute that into the remaining corners for establishing equality in the system [6]. The system itself is recognized as the PE where more than one PE can be presented in one system. When the work-stealing process used to have a big role play in the whole system design, then it is known as the distributed system as all the tasks are distributed in equal formats. Besides this, SLL is a type of this system which is focused on overheads of offloading minimal tasks. Here the large tasks are not considered for the remote PEs effectively. The policy of this system is quite similar to the FCFS policy [7].

3.3. LLL

The LLL is considered as the Large LargeLarge system. The model itself defines that it is associated with large tasks only. Therefore the system faces a lot of greedy approaches, such as executing the larger task in a short amount of time and some of the others. Besides this, the system is also having a lot of similar tasks from different PEs [8]. There, an idle PE is present which is only responsible to handle this type of large task. It takes up almost ⅕ th of the overall CPU cores while performing the functions effectively. During the execution of the large tasks, the performance of the CPU becomes lower in other areas, therefore it pauses the execution of smaller tasks to provide the best efficiency to the larger task execution cores [9].

3.4. LLS

The LLS stands for Large Large Small. This LLS system is mainly focused on large tasks whereas it is also focused on small tasks as well. The LLS mainly helps to take the large tasks for the same clusters and same pre-installation environment whereas the other clusters are also used but in a remote way for having a better execution process of the small tasks. Generally, the larger task execution starts first due with the help of the nearby victim. During the process of the larger task execution, another remote PE helps to execute the small tasks effectively. This system works only in particular sequences when the number of tasks is equal to the number PEs in the whole distributed system [10].

4. Comparison in the work-stealing processes

Table 1 compares the different work-stealing process that used for load balancing in a distributed system.

Table 1: Comparison of different work-stealing processes

Facts	SSL	SLL	LLL	LLS
Work Load	In the SSL the workload is having three segmentations. Where the small tasks get the highest priority to be executed. Large tasks are having the lowest priority.	In the SLL system, the workload is also divided into three segments. But here the small tasks have the highest priority of execution and after that, the logistics will get permission for execution [8].	In the LLL system, the workload is divided into three sections where each of the sections is responsible to execute the larger tasks only.	In the LLS system, the workload is also divided into triple segments where the larger tasks are having the highest priority for execution and the smaller tasks are also having the medium priority. During the execution of the larger task and extra PE helps to execute the smaller task at the same amount of time [8].
Priority	The smaller task has the highest priority in the overall processing.	The smaller task has the highest priority in the overall processing where the largest tasks also have permission to execute at the same time.	Largest tasks have the highest priority in the overall processing.	Although larger tasks have the highest priority of execution the smaller test is also executed separately with the help of other pre-installation environments.
Policy	The small small large system follows the basic format of FCFS policy.	The small large large system also follows the FCFS policy	Large LargeLarge does not follow the FCFS policy.	Large large and small systems are also associated with the FCFS policy.

Algorithms	In this scenario, the remote cluster level selects the largest tasks.	In this particular scenario, the source does not have any importance at all. It can be any cluster having the same identity or different from each other.	LLL system does not use any type of remote clusters for the ultimate execution [8].	LLS uses the remote cluster for selecting the smallest task to execute at the same time when the largest tasks are getting executed. To support this execution it uses an extra pre-installation environment present nearby the victim.
The latency of the network	SSL system does use the latency network to have a significant gap between two theft attempts and have a single victim based task [11].	SLL does not use the latency network.	The LLL system does not use the latency network.	LLS system the over-saturated network or bisect network where two to three PEs work in a single time.

5. Conclusion

Heavy-duty performances are very much observed in the current date in every circumstance, especially in the IT industries. Therefore parallel computing and some other topologies come into the scene. The distributed system is considered when the multiple functions are needed to be executed at the same time. It generally helps to divide the overall tasks into two segments one is the largest tasks and the other one is the smaller task. The systems defined above are the four combinations of the overall workflow system that are used in terms of managing all the tasks. Each of those has separate duties in separate circumstances. To have a greater idea about the work-stealing process in short but very deeply, it is recommended to follow the whole research effectively.

References

[1] *A. Lasserre, R. Namyst, and P. W. Easypap, "EASYPAP : a Framework for Learning Parallel Programming," 2020.*

[2] *G. E. Blelloch, J. T. Fineman, Y. Gu, and Y. Sun, "Optimal Parallel Algorithms in the Binary-Forking Model," in Proceedings of the 32nd ACM Symposium on Parallelism in Algorithms and Architectures, 2020, pp. 89–102.*

[3] *S. Kehrer and W. Blochinger, Equilibrium : an elasticity controller for parallel tree search in the cloud, no. 0123456789. Springer US, 2020.*

[4] *R. Carratalá-sáez, M. Faverge, G. Pichon, G. Sylvand, and E. Quintana-ortí, "Tiled Algorithms for Efficient Task-Parallel H-Matrix Solvers," in PDSEC 2020 - 21st IEEE International Workshop on Parallel and Distributed Scientific and Engineering Computing, 2020, pp. 1–10.*

[5] *K. Singer, K. Agrawal, and I. A. Lee, "Scheduling I / O Latency-Hiding Futures in Task-Parallel Platforms ∗," in Symposium on Algorithmic Principles of Computer Systems, 2020, pp. 1–15.*

[6] *J. Zhang, C. Yang, Y. Li, L. Chen, and X. Yuan, "LBVis : Interactive Dynamic Load Balancing Visualization for Parallel Particle Tracing," in 2020 IEEE Pacific Visualization Symposium, 2020, pp. 91–95.*

[7] *V. Freitas et al., "PackStealLB : A Scalable Distributed Load Balancer based on Work Stealing and Workload Discretization," 2020.*

[8] *M. Wang, T. Ta, L. Cheng, and C. Batten, "Efficiently Supporting Dynamic Task Parallelism on Heterogeneous Cache-Coherent Systems," in 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA) Efficiently, 2020, pp. 173–186.*

[9] *F. Fritz, M. Schmid, and J. Mottok, "Accelerating Real-Time Applications with Predictable Work-Stealing," in Architecture of Computing Systems – ARCS 2020, 2020, pp. 241–255.*

[10] *T. Wenjie, Y. Yiping, L. Tianlin, S. Xiao, and Z. Feng, "An Adaptive Persistence and Work-stealing Combined Algorithm for Load Balancing on Parallel Discrete Event Simulation," ACMTrans. Model. Comput. Simul, vol. 30, no. 2, pp. 1–26, 2020.*

[11] *P. Leca, W. Suijlen, L. Henrio, and F. Baude, "Distributed futures for efficient data transfer between parallel processes," in Proceedings of the 35th Annual ACM Symposium on Applied Computing, 2020, pp. 1344–1347.*