World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:2, No:7, 2008

Time Comparative Simulator for Distributed Process Scheduling Algorithms

Nazleeni Samiha Haron, Anang Hudaya Muhamad Amin, Mohd Hilmi Hasan, Izzatdin Abdul Aziz, and Wirdhayu Mohd Wahid

***Abstract*—**In any distributed systems, process scheduling plays a vital role in determining the efficiency of the system. Process scheduling algorithms are used to ensure that the components of the system would be able to maximize its utilization and able to complete all the processes assigned in a specified period of time. This paper focuses on the development of comparative simulator for distributed process scheduling algorithms. The objectives of the works that have been carried out include the development of the comparative simulator, as well as to implement a comparative study between three distributed process scheduling algorithms; *sender-initiated*, *receiver-initiated* and *hybrid sender-receiver-initiated* algorithms. The comparative study was done based on the Average Waiting Time (AWT) and Average Turnaround Time (ATT) of the processes involved. The simulation results show that the performance of the algorithms depends on the number of nodes in the system.

***Keywords*—**Distributed Systems, Load Sharing, Process Scheduling, AWT and ATT.

I. Introduction

SCHEDULING plays an important role in distributed systems in which it enhances overall system performance metrics such as process completion time and processor utilization [1]. From the research that has been carried out, there are two main distributed process scheduling algorithm, namely the *sender-initiated* and the *receiver-initiated* algorithm [2]. The third algorithm which is a *hybrid sender-receiver* algorithm as said to be the solution to overcome the problem from the two algorithms [3]. The performance of all three algorithms is benchmarked using the software simulator developed in this project.

The basic idea behind distributed process scheduling is the same as normal scheduling, which is to enhance overall system performance metrics [4]. However, in distributed systems the existence of multiple processing nodes poses a challenging problem for scheduling processes onto processors, and vice versa. This is because scheduling is not only done locally, but globally across the system. Process created at one node can migrate to other nodes in the system to redistribute work load as to achieve improved system performance. One of the main uses for global scheduling is to perform load sharing between processors. Load sharing allows busy processors to load some of their work to less busy, or even idle, processors [5].

Load balancing is a special case of load sharing, in which the goal of the global scheduling algorithm is to keep the load even (or balanced) across all processors [6]. Sender-initiated load sharing occurs when busy processors try to find idle processors to load some work. Receiver-initiated load sharing occurs when idle processors seek busy processors [7]. It is now accepted wisdom that, while load sharing is worthwhile, load balancing is generally not worth the extra effort, as the small gain in execution time of the tasks is more than offset by the effort expended in maintaining the balanced load.

Within the distributed system each individual node has its own policy for deciding when to accept or remove tasks. The characteristics of the distributed scheduling algorithm are normally depended on the reason of its existence such as information exchange, resource sharing, and increased reliability through replication and increased performance through parallelization [5]. A scheduling policy may be thought of as having four distinct parts: the transfer policy, the selection policy, the location policy, and the information policy. The transfer policy decides *when* a node should migrate a particular task, and the selection policy decides *which* task to migrate. The location policy determines a partner node for the task migration, and the information policy triggers and contains the collection of system state form all nodes: when, what and where [8].

Scheduling algorithms themselves can also be characterized as being either static or dynamic [1]. The decisions are based on both process characteristics and the current state of the system. A static approach calculates (or pre-determines) schedules for the system. It requires prior knowledge of a process's characteristics and requires little run-time overhead. By comparison, a dynamic method determines schedules at run-time thereby furnishing a more flexible system that can

Manuscript received April 20 , 2006.

Nazleeni Samiha Haron, Anang Hudaya Muhamad Amin, Mohd Hilmi Hasan and Izzatdin Abdul Aziz, are now with Information and Communication Technology / Business Information Systems (ICT/BIS) Programme of Universiti Teknologi Petronas, Perak, Malaysia. (e-mails : nazleeni@petronas.com.my, ananghudaya@petronas.com.my , mhilmi_hasan@petronas.com.my and izzatdin@petronas.com.my).

Wirdhayu Mohd Wahid was a student of Information and Communication Technology / Business Information Systems (ICT/BIS) Programme of Universiti Teknologi Petronas, Perak, Malaysia. She is now with Konsortium Jaya Sdn, Bhd, Shah Alam, Selangor, Malaysia. (e-mail : wmw_ayu83@yahoo.com).

deal with non-predicted events. It has higher run-time cost but can give greater processor utilization. In this paper, the authors focus on the dynamic process scheduling algorithms because it operates on load distribution strategy that is useful in a system where the primary performance goal is in maximizing of processors utilization as opposed to the minimization of runtime for individual jobs [1][8].

In this project, we define a distributed system as the interconnected collection of autonomous computers, processes, or processors, which are referred to as 'nodes'. To qualify as "autonomous', the nodes must at least be equipped with their own private control, thus a parallel computer of single-instruction does not qualify as distributed system. To qualify as 'interconnected', the nodes must be able to exchange information via network. A node contains processor(s) and processor support hardware. This is termed the internal environment. Also, each node has an interface to its external environment. The external environment of a node can contain other nodes (connected via an arbitrary network) and interfaces to external sensors, actuators or other controllable devices. The nodes in a system interact to achieve a common objective [9]

The objectives of this project are twofold. Firstly, is to conduct a comparative study among the three scheduling algorithms, which *are sender-initiated*, *receiver-initiated algorithms* and *hybrid sender-receiver algorithm*. From the research, the characteristics of all algorithms and their merits will be revealed. In order to prove this, the second objective of this project is to design and build the comparative simulation application that would be able to run and test each of the algorithms mentioned.

The paper is organized in the following sections. Section 2 will describe more on the related works that have been carried out by other people, and comparing those with the authors' works. Section 3 will give a brief overview of the distributed process scheduling algorithms. Section 4 describes the related issues in the simulator development that has been carried out in this project. Section 5 provides the detailed description of the results obtained and some discussion on the related issues being focused on. Finally, Section 6 provides conclusion to this paper.

II. Related Works

Load sharing is essential in ensuring the smooth performance of distributed systems. Therefore, considerable efforts have been put in studying this topic using simulation, prototypes and analytic models evaluating various performance metrics. Most of the researchers [10-17] focus on comparing between sender-initiated and receiver-initiated scheduling algorithms and only a few studies [18-22] have considered the hybrid of sender-and-receiver-initiated - in addition to the sender-initiated and receiver-initiated algorithms.

Comparison has been made by [1] to evaluate the performance between sender-initiated policy and receiver-initiated policy in terms of system workload. Their results prove that sender-initiated policy is better than receiver-initiated policy in light to moderate system loads while receiver-initiated policy is better than sender-initiated policy in high system loads. In addition, [3] and [4] have conducted a study towards the performance of sender-initiated and receiver-initiated policies in both homogenous and heterogeneous distributed system with regards to First Come First Serve (FCFS) and Round Robin (RR) scheduling policies. Apart from that, the study also includes the impact of variance in job service times and inter-arrival times. [5] provides the explanation on performance sensitivity of the sender-initiated and. receiver-initiated policies, to three factors: node-scheduling policy, variance in job inter-arrival, while [8] has reported the performance of several load sharing policies based on their implementation of both sender initiated and receiver initiated policies on a five node system connected by a 10Mbps communication network. The major difference that has been adopted by the authors in this study, as compared to the previous related works is such that the study focuses on evaluating the distributed scheduling algorithms using the Average Waiting Time (AWT) and Average Turnaround Time (ATT). AWT and AWT are believed to be the important factors to indicate the efficiency of any distributed systems.

On the other hand, [7] has conducted a study and compared the sender-initiated, receiver-initiated and hybrid (it is called symmetrical-initiated in this literature) policies pertaining to system workload and the effect of probing to overall system performance. [12] has also suggested a hybrid algorithm to overcome the limitations of both sender-initiated and receiver-initiated algorithms. In this research, the authors are focusing on evaluating the performance of three scheduling algorithms with regards to the AWT and ATT values obtained from the simulation.

III. Scheduling Algorithms

Scheduling plays an important role in distributed systems in which it enhances overall system performance metrics such as process completion time and processor utilization [8]. It will increase speed of the execution of the workload and executed more quickly with having the scheduling algorithm [2].

We will study three scheduling algorithms, which are *sender-initiated algorithm*, *receiver-initiated algorithm*, and *hybrid sender-receiver algorithm.* There are two *triggers* that would initiate them – the creation of a new process and when a process in queue finishes execution. In this study, we also consider the threshold-based transfer policy to determine when a processor should migrate a task or request for a task. The queue length will be used as the indicator for the load.

A. Sender-Initiated Algorithm

For *sender-initiated algorithm*, there is a *queue threshold* (ST), which can also be referred to as the *maximum queue*

threshold. When the queue length exceeds that of the maximum queue threshold, the *sender-initiated algorithm* is initiated. Fig. 1 shows the representation of *sender-initiated algorithm* node's queue.

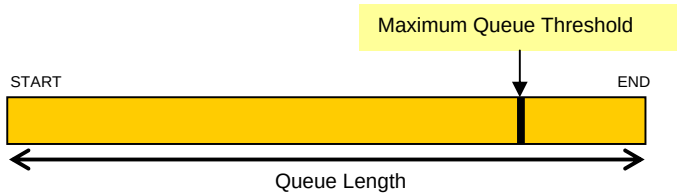

Fig. 1 Representation of a Sender-initiated Algorithm Node's Queue

B. Receiver-Initiated Algorithm

For *receiver-initiated algorithm*, the *queue threshold* (ST) is also known as the *minimum queue threshold*. If the queue length falls below this threshold, the *receiver-initiated algorithm* is then initiated. Fig. 2 shows the representation of a *receiver-initiated algorithm* node's queue.

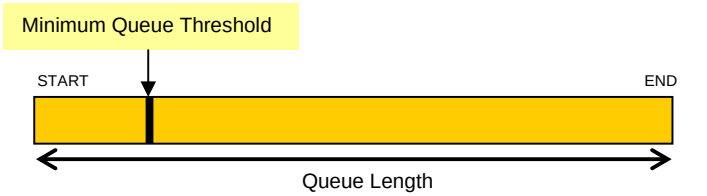

Fig. 2 Representation of a Receiver-initiated Algorithm Node's Queue

C. Hybrid Sender-Receiver-Initiated Algorithm

A *hybrid sender-receiver node* is one where both the *sender-initiated algorithm* and *receiver-initiated* algorithm is implemented. For this, the node would require two different *threshold* values. One would define the maximum queue length before the *sender-initiated algorithm* is executed and the other the minimum queue length for the *receiver-initiated algorithm*. As with the previous two algorithms, two events trigger the hybrid algorithm – the arrival of a new process and after the execution of a process. Fig. 3 shows the representation of this hybrid algorithm.

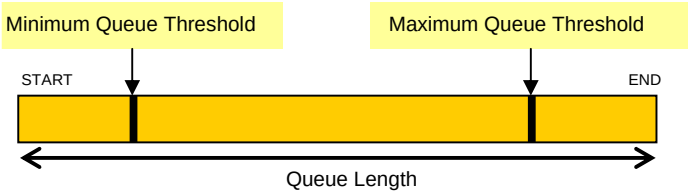

Fig. 3 Representation of a Hybrid Sender-Receiver Node's Queue

D. Performance Metrics

The simulator developed allows user to benchmark the performance of all three algorithms in a controlled environment based on AWT and ATT. The lower the ATT and AWT, the better the performance. ATT and AWT are said to be the best metrics to compare the performance of the algorithms [1].AWT and ATT are computed as follows :

AWT = [(time process in system - service time)/n]
ATT = [(waiting + service times)/n]
where n is number of nodes.

IV. Simulator Development

A. Use Case Design

Fig. 4 shows the system from the viewpoint of the user, with user designated as an actor. User is able to perform the following functions: control the simulator, change the simulation parameters and display results. Simulation control involves starting and stopping the simulator. Change simulation parameters allows for the user to change several parameters of the simulator including simulation type, number of nodes, arrival and service time of processes, and simulation run time. The last function is display results which, taking the results of the previously ran test or a previously saved simulation session, shows a comparative display between the tests.

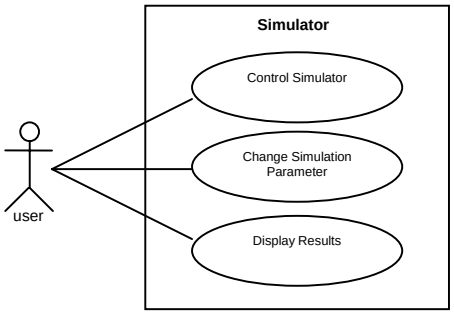

Fig. 4 Use Case Diagram (User's Viewpoint)

Fig. 5 use-case diagram shows the system from the viewpoint of the simulator itself. There is only one actor involved, which is the Simulator. Simulator is the main entity, and encompasses the overall control of the system. It controls the simulation, gather simulation parameters, store statistical data, and shows output to the user.

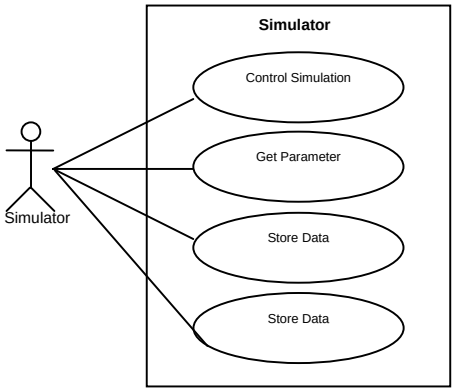

Fig. 5 Use Case Diagram (System's Viewpoint)

B. Implementation

The simulator that has been developed is a Java-based application that runs on Java Virtual Machine (JVM). The advantage of having Java as a development platform is such that it would be able to cater for all operating systems, having JVM properly installed.

Before simulating the scheduling algorithms, the simulator needs to be configured accordingly. The simulation setting requires the user to enter the number of nodes, which represents the processor and to specify the number of generated process randomly. There are two modes of running the simulation; single or comparative. Single simulation allows us to see any one of the three algorithms' performance while a comparative simulation runs the three algorithms one after another. Both simulation types output the results in a graph (based on average turnaround time and average waiting time).

When the simulator has started, the load sharing mechanism of the processes will be shown. Fig. 6 shows how load sharing is performed by each node depending on the respective scheduling algorithm.

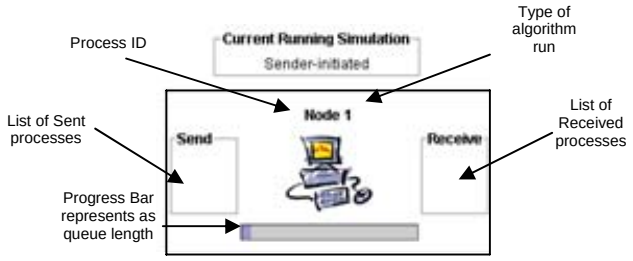

Fig. 6 Representation of a Node in terms of Load Sharing

Fig. 7 shows the simulator's message window that displays the real-time contextual data. It provides a textual display of the running of the simulation such as the addition of a process, the execution of a process, and the migration of a process are displayed here. There are two buttons available at the bottom of the window, the Close and Save button. Pressing Close will close the Message Window, while pressing Save will save the output of the Message Window to a file.

Fig. 7 Message Window

Popup window will appear right after the simulation processes are completed and after all the nodes involved are terminated. The window exhibits the results of the simulation in two graphs of which each represents AWT and ATT respectively. The graphs show plotted line of AWT and ATT versus time.

V. Results & Discussion

In achieving the first objective of this project, the authors have implemented three different test cases (see Table I), and all the three scheduling algorithms; *sender-initiated*, *receiver-initiated*, and *hybrid sender-receiver-initiated* are tested against these test cases. The test cases differ in terms of the number of nodes, which represents the system size. This parameter was chosen due to the fact that an important feature of any load sharing algorithm is that the performance should maintain although the number of processors in the system increases [23].

TABLE I
Test Cases

Test Case	Number of Node
1	2
2	10
3	50

In order to standardize the experiment and to ensure comparable results, the authors have chosen a few parameters to be invariant as shown in Table 2. The values chosen are actually being determined from experimentation as well as from the authors' knowledge and are believed can produce good comparable results .

TABLE II
Invariant Values of the System Parameter

Parameter	Values
No. of Generated Process	50 - 100
Queue Length	35
Upper Threshold	25
Lower Threshold	10
Process Arrival Time	200-600 ms
Process Burst Time	50-1000 ms

A. Test Case 1: Simulation using Two Nodes

For this test case, two nodes is selected as the parameter because we want to see the performance of all the three algorithms in the minimum scenario. Fig. 8 shows the results of the simulation of distributed process scheduling by using two nodes. It should be noted that the results shown that hybrid sender-receiver-initiated algorithm executes more time to complete all the processes generated, as compared to both sender-initiated and receiver-initiated algorithms. This might be due to the needs for the algorithm to evaluate both minimum and maximum threshold values before any processes could be executed. Sender-initiated has the lowest

AWT and ATT and receiver-initiated has the highest AWT and ATT values. This results has proven the claim of [10] that under light loads (since only two nodes involved), the sender-initiated algorithm will work well as compared to receiver-initiated. The hybrid sender-receiver-initiated algorithm is expected to have the identical or close ATT and AWT values to sender-initiated algorithm. However, as shown in the results, it does not perform well but instead it sits in the middle of other two algorithms. This issue may exist due to the small number of nodes used.

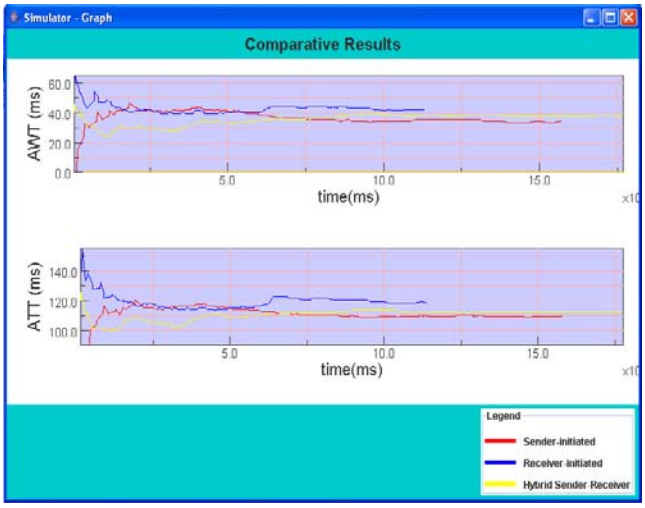

Fig. 8 Graphs for Comparative Simulation in Two Nodes Environment

B. *Test Case 2: Simulation using 10 Nodes*

In order to demonstrate the performance of the algorithms in a medium scale system, the authors choose to vary the system size to 10 nodes. Fig. 9 shows the results of the simulation by using 10 nodes. The results gained from this comparative simulation show that all the algorithms achieved almost similar ATT and AWT values.. However, the results indicated that both sender-initiated and hybrid sender-receiver-initiated algorithms incurred longer processing time. This is true, since both algorithms required time to wait for any processes to be placed in the queue.

C. *Test case 3: Simulation using 50 Nodes*

Fig. 10 shows the results of the simulation of distributed process scheduling by using 50 nodes, which is the heaviest and largest simulated system size. As compared to the previous test cases, the results show a slight increased in both ATT and AWT values for all algorithms implemented. However, the ATT and AWT values are comparatively similar to the values obtained from test case 2. This simply means that there are no significant differences in the AWT and ATT values for all algorithms.

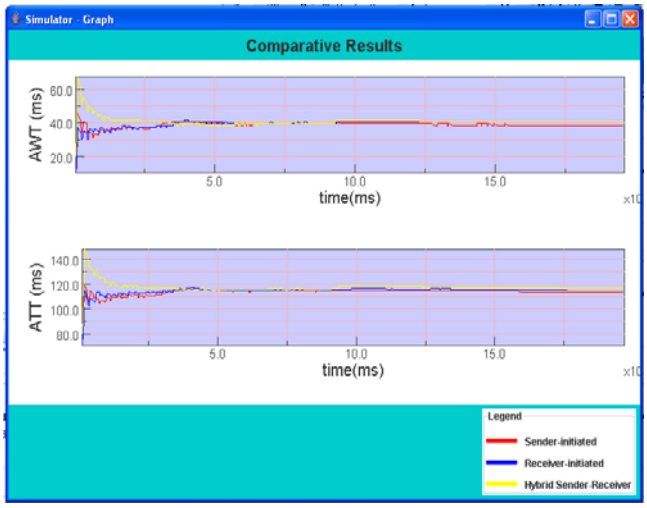

Fig. 9 Graphs for Comparative Simulation in 10 Nodes Environment

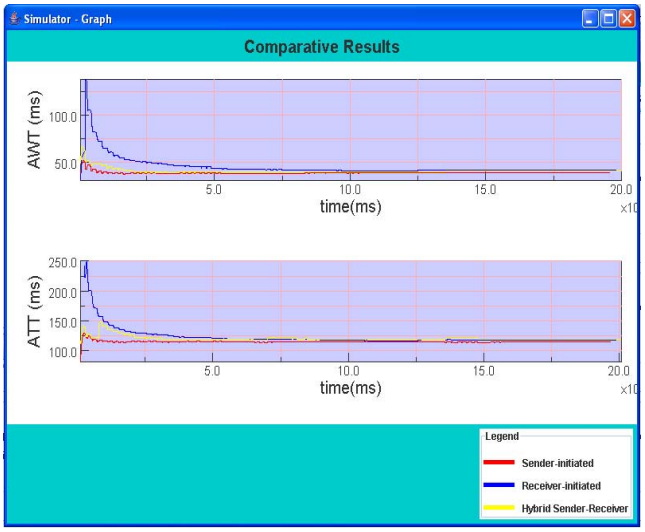

Fig. 10 Graphs for Comparative Simulation in 50 Nodes Environment

From the test cases that have been carried out, it seems that the number of nodes used in a single processing environment plays an important role in determining the performance of the three algorithms. The justification is such that the bigger the number of nodes is being used; the graphs indicated that the ATT and AWT values for each of the algorithms being implemented are almost alike. The results prove the works of [23-24] that the performance of the algorithms becomes insensitive to the number of nodes as the number of the nodes increases.

VI. Conclusion

The simulator developed allows user to benchmark the performance of all three algorithms in a controlled environment based on AWT and ATT. The expectations that the hybrid algorithm generally works better are not true for certain cases. From this project, it has been known that the

number of nodes participated in a processing environment also plays an important role in determining the efficiency of the system.

Even though the software simulator is completed, a lot of improvement can be done to make it work better. Currently the simulator emulates distributed processors as threads. This method is inferior to actually running simulation software on multiple processors. The use of sockets and Remote Method Invocation (RMI) can increase the accuracy of the simulator and delivers more realistic results. The software can also be improved by allowing the user to finely adjust each and every parameters of the simulator. This was not implemented since the major parameters are already present and was thought to be enough for the simulator to work well.

Additional parameters at this time can be considered trivial but would make a worthy addition in the future. Currently the statistical output of the simulator is very much textual in nature except for the graph presented. It is hoped that the simulator's output can be further enhance either by utilizing more visual elements or using extensible markup language (XML) to represent the data. XML allows the data to be used in a variety of ways, either as an input for other programs/parsers or for output purposes.

REFERENCES

[1] Tel, G., Introduction to Distributed Process Scheduling. 1998, University of Cambridge..
[2] Chow, R. and T. Johnson, Distributed Operating Systems and Algorithms. 1997: Addison-Wesley.
[3] Ramamritham, K. and J.A. Stankovic, Dynamic Task Scheduling in Hard Real-Time Distributed Systems. IEEE Software, 2002. 1(3): p. 65-75.
[4] Audsley, N. and A. Burns, Real -Time Scheduling, in Department of Computer Science. 1994, University of York.
[5] Boger, M., Java in Distributed Systems. 2001: Wiley.
[6] Malik, S., Dynamic Load Balancing in a Network Workstations. 2003: Prentice-Hall.
[7] Stankovic, J.A., Simulations of three adaptive, decentralized controlled, job scheduling algorithms. Computer Networks, 1999. 8(3): p. 199-217.
[8] Chaptin, S.J., Distributed and Multiprocessor Scheduling. 2003, University of Minnesota.
[9] Audsley, N., Scheduling Hard Real-Time Systems. 2000.
[10] Eager, D.L., E.D. Lazowska, and J. Zahorjan, A Comparison of Receiver-Initiated and Sender-Initiated Adaptive Load Sharing. 1985, University of Washington.
[11] Dandamudi, S.P. The Effect of Scheduling Discipline on Dynamic Load Sharing in Heterogeneous Distributed Systems. in IEEE Int. Symp. on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS). 1997. Haifa, Israel.
[12] Dandamudi, S.P., The Effect of Scheduling Discipline on Sender-Initiated and Receiver-Initiated Adaptive Load Sharing in Homogeneous Distributed Systems. 1995, Carleton University.
[13] Dandamudi, S.P., Sensitivity evaluation of dynamic load sharing in distributed systems. Concurrency, IEEE, 1998. Volume 6(3): p. 62 - 72.
[14] Dikshit, P., S.K. Tripathi, and P. Jalote, SAHAYOG: A Test Bed for Evaluating Dynamic Load--Sharing Policies. Software -- Practise and Experience, 1989. 5: p. 411-435.
[15] Svensson, A. Dynamic alternation between receiver-initiated and sender-initiated load sharing. in International Conference on Databases, Parallel Architectures and Their Applications (PARBASE-90). 1990.
[16] Dandamudi, S.P. and H. Hadavi. Performance Impact of I/O on Sender Initiated and Receiver Initiated Adaptive Load Sharing in Distributed Systems. in Int. Conf. Parallel and Distributed Computing Systems. 1996. Dijon, France.
[17] Kremien, O. and J. Kramer, Methodical analysis of adaptive load sharing algorithms. Parallel and Distributed Systems, IEEE Transactions on, 1992. 3(6): p. 747-760.
[18] Shivaratri, N.G. and P. Krueger. Two Adaptive Location Policies for Global Scheduling Algorithms. in IEEE International Conference on Distributed Computing Systems (ICDCS). 1990.
[19] Krueger, P., Distributed Scheduling for a Changing Environment. 1988, University of Wisconsin-Madison.
[20] Mirchandaney, R., D. Towsley, and J.A. Stankovic, Analysis of the effects of delays on load sharing. Computers, IEEE Transactions on, 1989. 38(11): p. 1513 - 1525.
[21] Antonis, K., J.D. Garofalakis, and P.G. Spirakis. A Competitive Symmetrical Transfer Policy for Load Sharing. in Proceedings of the 4th International Euro-Par Conference on Parallel Processing. 1998.
[22] Dasgupta, P., A.K. Majumder, and P. Bhattacharya, V_THR: An Adaptive Load--Balancing Algorithm. Journal of Parallel and Distributed Computing. 42(2): p. 101-108.
[23] K. Benmohammed-Mahieddine and P. M. Dew, A Periodic Symmetrically-Initiated Load Balancing Algorithm for Distributed Systems, ACM SIGOPS Operating Systems Review, Vol. 28, No. 1, 1994, pp. 66--79.
[24] Y.T. Wang and R.J.T. Morris, Load Sharing in Distributed Systems, IEEE Transactions on Computers C-34(3)pp. 204-217 (March 1985).