Volume 6, Issue 2, February 2016 ISSN: 2277 128X
International Journal of Advanced Research in Computer Science and Software Engineering
Research Paper
Available online at: www.ijarcsse.com

Simulated Analysis of Static Policies in Multiprocessor Environment

Sukhpreet Kaur*, Harpreet Kaur
Faculty, Department of CSE PUSSGRC, Hoshiarpur,
Punjab, India

Abstract— Parallel machines use space-sharing strategies to handle multiple jobs at the same time by dedicating the nodes to a single job until it completes. Performance has been an important factor to find out the desired results of parallel system. The objective of maximizing the performance of the system has led to the growth of the job schedulers in parallel environment that match the requirements and workload with resource availability in terms of basic architecture and processors. This paper gives a overview of some static space sharing policies which are FCFS (First come first served), LJF (Largest job first) and FPFS (Fit processor first served) in which a job scheduler searches jobs in the job queue and dispatches a job that fits to idle processors. These policies are implemented using simulation. The performance of all the policies are analyzed and compared on the basis of performance metrics average waiting time, mean response time and utilization.

Keywords—Job Scheduling, Rigid Jobs, Space Sharing, Response Time.

I. INTRODUCTION

Many applications today require more computing power, faster results which can't be obtained with a traditional sequential computer. Instead of using uniprocessor systems, multiprocessor systems or an array of parallel processing elements controlled by one uniprocessor are being used to solve large problems. There is high demand for fast and efficient computers in many applications, where large scale computations are performed. These applications include natural sciences, database, military and defense, real time and for commercial purposes. Parallel processing computers are needed to meet these demands. The workload can be shared between different processors. This results in much higher computing power and performance than it could be achieved with traditional single processor system.

Time-sharing refers to any scheduling approach whereby when jobs of higher priority comes then threads can be preempted during execution and resumed later. The number of jobs that each processor can execute concurrently is known as the multiprogramming level. On the other hand, space-sharing approach provide a thread exclusive use of a processor until its execution is complete or a maximum time limit has been reached and the thread is terminated. Interactive jobs that require low latency are usually executed using time-sharing, while batch jobs that require undisturbed performance are executed on dedicated processors using space-sharing. As such, with a submission time a job is inherently associated and based on the current state of an ordered job stream scheduling algorithms must make online scheduling decisions.

Parallel space-sharing job scheduling algorithm tends to play two roles one is selecting a job from the set of competing jobs as well as allocating processors to the job from the free processors . Space sharing [3] is classified into partitions:

- **Fixed:** The size of the partition is fixed throughout the lifetime of the job and is defined by the system administrator and only reboot can modify it.
- **Variable:** The size of the partition is determined at the time when the job is submitted upon request of user.
- **Adaptive:** The size of the partition is determined at the time when the job is initiated by the scheduler, based on the system load, and by taking into account the request of the user.
- **Dynamic:** The size of the partition may change during the execution of a job, to reflect changing requirements and depends on system load.

It is necessary to provide effective scheduling strategies to meet the desired quality of service parameters from both user and system perspectives. Specifically, the goal of the system is to reduce response and wait times for a job, maximize the utilization of the system, and be fair to all jobs regardless of their size or execution times. Job scheduling is a discipline whose purpose is to decide when and where each job should be executed from the perspective of the system. [3] *A job* to refer to some parallel program, composed of multiple concurrent *threads*, submitted to the system for execution. [3] As such, a job is inherently associated with a submission time and scheduling algorithms must make *online* scheduling decisions based on the current state of an ordered job stream. Each job is characterized along two dimensions: its *length* as measured by execution time and its *width* or *size* as measured by the number of threads. Job sizes are not necessarily fixed before or during execution, but are often so in practice.

The applications are characterized as follows [8]:

- **Rigid jobs:** The number of processors which are assigned to a job is specified by the user and is external to the scheduler, and exactly that number of processors is made available to the job throughout its lifetime for execution.
- **Moldable jobs:** The number of processors assigned to a job is determined by the scheduler within certain constraints i.e. when the job is first activated, and it uses same number of processors throughout its execution.
- **Evolving jobs:** The job goes through different levels that may require different numbers of processors, so the number of processors allocated may change during the execution of job in response to the job requesting more processors or releasing some. Each job is allocated at least the number of processors it requires at each point in its execution but it may be allocated more to avoid the overheads of reallocation at each phase.
- **Malleable jobs:** The number of processors assigned to a job may change during the job's execution, as a result of the system giving it additional processors or requiring that the job release some. The manner in which an application is written determines which of the four types is used.

An important goal of any parallel-system scheduler is to promote the productivity of its users. The scheduler has to keep its users satisfied to achieve high productivity and encourage them to submit more jobs. Due to the high costs involved in deploying a new scheduler, it is not common to experiment with new designs in reality for the first time. Instead, whenever a new scheduler is proposed, it is first evaluated in simulation, and only if it demonstrates significant improvements in performance, it then becomes applicable for an actual use. The role of simulations is thus critical for the choices made in reality. As simulator provides a powerful way to predict performance before the system under study has not been implemented. The algorithms are evaluated through simulation. Some of the parameters of the workload are modelled, the most significant of which are job interarrival time, job parallelism and job runtime. The job interarrival times and job service times are hyper exponential distributed. The job parallelism is calculated through uniform distribution.

The basic scheduling problem can be stated as follows:

1. There are multiple jobs in parallel system, and efficient scheduling policy is used to allocate all jobs to the given free number of processors.
2. The performance of job scheduling algorithms may be influenced by several input factors such as the job sizes of all the jobs in the workload, the arrival patterns and job service times, type of distributions used for calculating arrival patterns and job service times and even on the type of scheduling algorithm being used.
3. Then the metrics like average wait time, mean response time and processor utilization are used to compare different scheduling algorithms.

II. METHODOLOGY

In order to fulfill this objective the simulator should be developed using .NET component framework. Simulator takes job id, the job arrival time, service time, number of processors required as its input and using them evaluates the performance of the selected scheduling algorithm.

A. Steps to be followed for performance evaluation

1) The arrival time and service time is generated according to hyper exponential distribution.
2) The job size is calculated through uniform distribution.
3) Follow the queuing system to arrange the process in particular order depending upon arrival time, and following the particular queue discipline like FCFS, LJF, FPFS etc.
Obtain results in form of start time of queued jobs, processor allocated to jobs and completion time of jobs.

III. STATIC SPACE SHARING TECHNIQUES

The simplest way to schedule a parallel system is with a queue.
A. FCFS: Under this algorithm, jobs are considered in the order of their arrival. If there are enough processors available to run a job, the processors are allocated and the job is started. But if enough processors are not available, the first job must wait for some currently running job to terminate and free additional processors. All subsequent jobs also wait so as not to violate the FCFS order. [5]
B. SJF (Shortest Job First): In this scheme the shortest job in the queue is executed first. The scheduler searches the entire queue first to find the next shortest job in the queue.
C. LJF (Largest Job First): The largest job in the queue is executed first .The scheduler searches the entire queue to find the next largest job in the queue.
D. FPFS: In FPFS(Fit processors first served) if there are not enough processors available for the front job in the job queue, then job scheduler searches the job queue for the job which fits first to number of processors available and consequently that job is dispatched and processors are allocated to the job. This algorithm is classified into FCFS/First-Fit, LJF/First-Fit and SJF/First-Fit.[1]
1) FCFS/First- Fit: A job scheduler searches jobs in a shared job queue form the top to the bottom and dispatches the first job with job size which is not greater than the number of idle processors.[7]
2) LJF/First- Fit: a job scheduler sorts the job in a shared job queue to the decreasing order of job size and then dispatches jobs in the same way as First-Fit.

3) SJF/First- Fit: it dispatches job in the same way as LJF/First- Fit except that a job scheduler sorts jobs to the increasing order of job size. It is same as SJF because of following reason. During searching jobs sorted by increasing order if scheduler does not find the job that fits the free processors then there is no job in the entire queue that fits free processors.[7]

E. Backfilling: Backfilling [2][6] algorithms allow small jobs from the back of the queue to execute before larger jobs that arrived earlier, thus utilizing the idle processors, while the latter will wait for enough processors to be freed. Backfilling is widely used technique in parallel job scheduling to increase system utilization and user satisfaction over conventional non-backfilling scheduling algorithms. It is different from traditional first-come-first-serve (FCFS) scheduling in the way that it requires the estimated running time of jobs to be known in advance before making scheduling decisions.

IV. PERFORMANCE METRICS

One problem with selecting a performance metric is that in a real system different metrics may be relevant for different jobs. For example, response time may be the most important metric for interactive jobs, while system utilization is more important for batch jobs. But in an open, on-line system, utilization is largely determined by the arrival process and the requirements of the jobs, not by the scheduler. This leaves response time as the main metric.

A Turnaround time measures the time between job_i's arrival and completion. [4]

$$Average\ Turnaround\ Time = \frac{\sum_{j\varepsilon j\ obs}(j.completion_{time} - j.arrival_{time})}{\sum_{j\varepsilon jobs}}$$

B. Utilization: Utilization is the most common system metric. In simulation studies, utilization simply is an indirect measure of makespan, as the workload of all schedulers is a constant.[4]

$$Utilization = \frac{\sum_{i=1}^{n} job_i\ .used\_processors\ * job_i\ .runtime}{Makespan * SystemSize}$$

C. Response time (RT) is defined as sum of waitingtime (Tw) and running time (Tr) and is often the metric of choice for system evaluation.

D. Slowdown= $\frac{Tw + Tr}{Tr}$

it evidently favors the short jobs.

E. Waitlimit is a deadline that a job continues waiting for being dispatched to processors, is given to each job.

Waitime= Job_i Start Time – Job_i Arrival Time

V. SIMULATION OF SPACE SHARING

In space sharing scheduling, a job requests a fixed number of nodes on arrival, and the scheduler finds and allocates the nodes, possibly after the job has been in a waiting queue. The job then executes on the assigned nodes until completion. The nodes are not shared and the jobs are never suspended during execution.

To simulate space sharing scheduling algorithms and optimizations, the only need to know is the arrival times and execution times of jobs. In most modern parallel machines, the physical location of the nodes on which a job executes does not significantly affect the execution, and hence the logged execution time can be taken as the execution time for simulations.

VI. WORKLOAD CHARACTERIZATION

Many studies have used workloads [9] consisting of a small number of parallel programs obtained from benchmark suites or industry, or synthetically generated workloads with somewhat arbitrary distributions. The major job parameters of interest are inter-arrival time, service time and degree of parallelism.

A. Job Inter-arriva1 Time: The most popular choice for modeling job arrivals has been a Poisson process. Many studies have used this distribution although have experimented with the hyper-exponential distribution.

An exponential distribution is known to have a CV (coefficient of variation) of 1. However, studies of workload traces from parallel computing sites have tended to find inter-arriva1 CV's considerably larger than 1.

B. Job Service time: Job service times have usually been modeled with a hyper-exponential distribution and workload studies tend to support this practice. The CV's from workload traces are generally in the range of 2-10, although numbers as high as 70.

It is always the query that whether job runtimes are correlated with job sizes. A weak positive correlation that larger jobs run longer has been reported by several authors but few researchers have modeled this.

Many studies have also noted the large numbers of small, short jobs often over half the total number of jobs. As well, they have al1 found that these small jobs consume a small fraction of the total resources.

C. Parallelism: The job size parameter has received the most widely varying treatment in the literature. This is due, not only to the lack of information concerning the true distribution, but as well to the difficulty of modeling the data that does exist. The reported data in workload studies are difficult to characterize briefly because they have a high discrete component, and tend to vary considerably from site to site.

VII. DISTRIBUTIONS USED

Distributions [10] of workload parameters typically have the property that the values are positive. There is no such thing as a negative runtime.

*A. The Exponential Distribution:*The exponential distribution (or rather, negative exponential distribution) is defined by the pdf

$f(x) = \frac{1}{\theta} e^{\frac{-x}{\theta}}$

and the CDF

$F(x) = 1 - e^{\frac{-x}{\theta}}$

An alternative form of the definition uses $\lambda = 1/\theta$ as the parameter. The pdf is then

$f(x) = \lambda e^{-\lambda x}$ and

the CDF is

$F(x) = 1 - e^{-\lambda x}$.

λ is interpreted as a rate parameter: it measures how many things happen per unit of x. The two forms are used interchangeably as convenient.

*B. The Hyper-Exponential Distribution:*The hyper-exponential distribution is obtained by selection from a mixture of several exponential distributions.

This means that each client either gets a service time from an exponential distribution with rate λ_1, which happens with probability p, or else it gets a service time from an exponential distribution with rate λ_2 (with probability 1 − p). Naturally, λ_1 should be different from λ_2. In the general case (with k stages) the pdf is

$f(x) = \sum_{i=1}^{k} p_i \lambda_i e^{-\lambda_i x}$

and the CDF

$F(x) = 1 - \sum_{i=1}^{k} p_i e^{-\lambda_i x}$

where $\sum_{i=1}^{k} p_i = 1$

VIII. DETAILS ABOUT WORKING OF THE SIMULATOR

1. GUI based – Easy to use & understand.

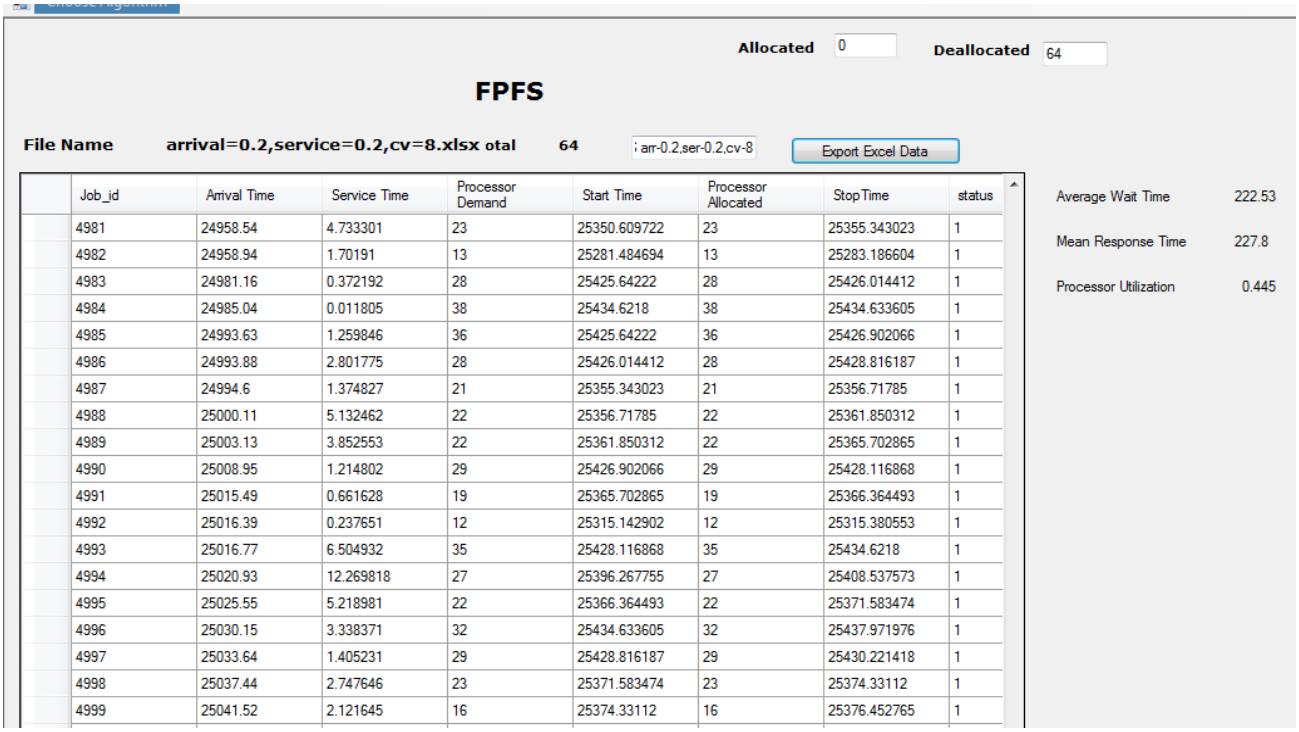
2. Data Backup- Data generated by simulation is stored in MS-Excel and can be used whenever needed.

3. Status – Simulator has shown the status of jobs i.e. whether it is completed i.e. 1 or running i.e. 0.

4. Various parameters shown in labels and text boxes of simulator- Various Parameters generated by Simulator after scheduling policy are being modeled in it are shown in various labels and text boxes.

5. Run-time performance measurement-It measures the completion time as well as average waiting time, mean response time and processor utilization during run-time.

6. Grid View- The grid view takes the input of arrival time, service time and processor demand from excel sheet and generates the output in the form of start time of job, processor allocated, stop time i.e. completion time of job and import back data to the excel sheet.



Job_id	Arrival Time	Service Time	Processor Demand	Start Time	Processor Allocated	StopTime	status
4981	24958.54	4.733301	23	25350.609722	23	25355.343023	1
4982	24958.94	1.70191	13	25281.484694	13	25283.186604	1
4983	24981.16	0.372192	28	25425.64222	28	25426.014412	1
4984	24985.04	0.011805	38	25434.6218	38	25434.633605	1
4985	24993.63	1.259846	36	25425.64222	36	25426.902066	1
4986	24993.88	2.801775	28	25426.014412	28	25428.816187	1
4987	24994.6	1.374827	21	25355.343023	21	25356.71785	1
4988	25000.11	5.132462	22	25356.71785	22	25361.850312	1
4989	25003.13	3.852553	22	25361.850312	22	25365.702865	1
4990	25008.95	1.214802	29	25426.902066	29	25428.116868	1
4991	25015.49	0.661628	19	25365.702865	19	25366.364493	1
4992	25016.39	0.237651	12	25315.142902	12	25315.380553	1
4993	25016.77	6.504932	35	25428.116868	35	25434.6218	1
4994	25020.93	12.269818	27	25396.267755	27	25408.537573	1
4995	25025.55	5.218981	22	25366.364493	22	25371.583474	1
4996	25030.15	3.338371	32	25434.633605	32	25437.971976	1
4997	25033.64	1.405231	29	25428.816187	29	25430.221418	1
4998	25037.44	2.747646	23	25371.583474	23	25374.33112	1
4999	25041.52	2.121645	16	25374.33112	16	25376.452765	1

Fig 1: Screenshot of working simulator.

Relative performance of the scheduling policies

In this section we compare the policies on the basis of performance metrics mean response time and utilization. The arrival rate for jobs is fixed i.e. 0.8 with CV_A-8. But the service rate varies with constant CV_S-7. And the MRT of FPFS is less as compared to both policies while utilization is more than both policies.

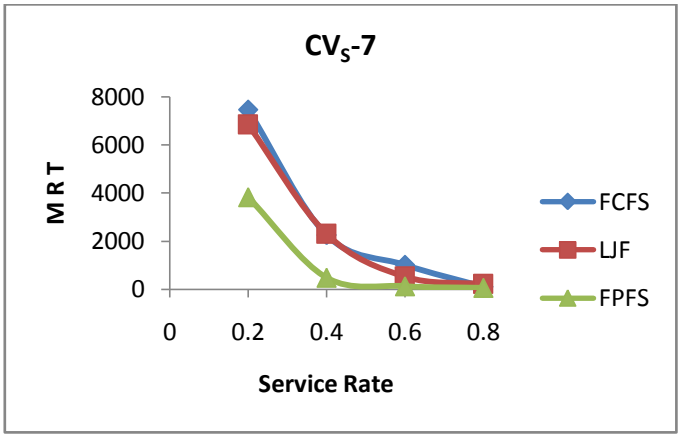


Fig 2: MRT with Service Rate Varies, CV_S –7, Arrival Rate – 0.8, CV_A-8

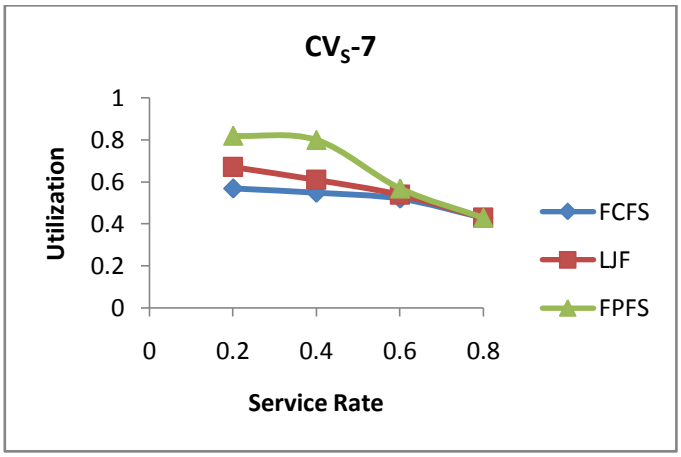


Fig 3: Utilization with Service Rate Varies, CV_S –7, Arrival Rate – 0.8, CV_A-8

IX. CONCLUSION

The objective of this paper is to evaluate several static space sharing techniques to enhance job scheduling for parallel systems. The several space sharing policies are evaluated on the basis of workload, scheduling policies, and performance evaluation methods and compared on the basis of performance metrics mean response time and utilization which concludes that FCFS gives the worst performance. The FPFS is more effective and more practical than LJF and FCFS space sharing techniques which search the job queue and dispatches jobs to free processors.

REFERENCES

[1] Kento Aida, Hironori Kasahara and Seinosuke Narita "Job scheduling scheme for pure space sharing among rigid jobs".

[2] Bo Li, Jun Chen, Man Yang, Erfei Wang "Impact of extending the runtime of underestimated jobs in backfilling schedulers", International Conference on Computer Science and Software Engineering, 2008.

[3] Jonathan Weinberg " Job Scheduling on Parallel Systems" University of California, San Diego 9500 Gilman Drive La Jolla, CA 92093-0505.

[4] Vitus J. Leung, Gerald Sabin, P. Sadayappan "Parallel Job Scheduling Policies to Improve Fairness: A Case Study", 39th International Conference on Parallel Processing Workshops,2010

[5] Dror G. Feitelson , Larry Rudolph, Uwe Schwiegelshohn "Parallel Job Scheduling — A Status Report".

[6] Barry G. Lawson, Evgenia Smirni, Daniela Puiu "Self-adapting Backfilling Scheduling for Parallel Systems".

[7] Kento Aida, "Effect of Job Size Characteristics on Job Scheduling Performance".

[8] D.G. Feitelson, L. Rudolph, U. Schwiegelshohn, K.C. Sevick and P. Wong, "Theory and practice in parallel job scheduling". In Job Scheduling Strategies for Parallel Processing D.G. Feitelson and L. Rudolph (Eds) , volume 1291 of Lecture Notes in Computer Science, pp. 1–34. Springer-Verlag, 1997.

[9] Gerard Lynch, "Parallel Job Scheduling on Heterogeneous Networks of Multiprocessor Workstations", Master Thesis University of Toronto, 1999.

[10] Dror G. Feitelson, "Workload Modeling for Computer Systems Performance Evaluation", Version 0.34, September 25, 2011.

