Scheduling Simulations: An Experimental Approach to Time-Sharing Multiprocessor Scheduling Schemes

Swinky Arora
Assistant Professor
Chandigarh University
Gharuan

Ankit Arora
Assistant Professor
Lala Lajpat Rai Institute of
Engineering and Tech. Moga

Gursharanjit Singh Cheema
Assistant Professor
Lala Lajpat Rai Institute of
Engineering and Tech. Moga

ABSTRACT
Real time systems that are logically programmed for scientific applications involve frequent job arrivals, thus requires a parallel architecture, so that maximum applications can be executed simultaneously resulting in less waiting time and maximum resource utilization. This must be achieved by workload partitioning & characterization, directs towards the development of Multiprocessor machines, a way to achieve parallel effects. Today, multiprocessor systems cover H/W replications that may replicates complete central processing units asynchronously or multiple executional units synchronously controlled by a different/common clock respectively. This research deals with the multiprocessor scheduling implemented via simulated time sharing environment containing logically programmed virtual processors and batch lists, each batch having its associated arrival time along with number of jobs where each job contains parameters such as Batch_id, Job_id and CPU Burst_time(defined as no. of cycles required) etc. The idea behind this theory is to distribute a number of simultaneously occurring jobs to virtual processor list corresponding to a scheduling algorithm. Synchronous architectures involve SIMD based model with data parallel aspects of computations, whereas Control parallel asynchronous MIMD machines are the future trends leading towards Instruction level parallel processors involving VLIW (very large instruction word) and superscalar machines.

General Terms
Multiprocessor Scheduling Policies & Mechanisms, Time Sharing & Space Sharing policies, Static vs Dynamic scheduling Decisions.

Keywords
Simulated Time-Sharing Environment, Job Distribution, Load Balancing, Workload Partitioning.

1. INTRODUCTION
Experimental studies and setups for parallel systems involve distributed clusters and simulated environments. Process Scheduling is the major pragmatic task over multiprocessor hardware. Scheduling over multiprocessor involves task placement and task adjustment. When a new job arrives and is actually placed inside processor queue according to some scheduling criteria or strategy, this is called task placement, whereas when already running jobs/tasks are reassigned or shifted among processors queue for the aim of load balancing is referred to as task adjustment [2]. Other aspects regarding scheduling decisions covers long-term also referred to as a global scheduling where scheduling decisions are made regarding how the jobs are to be allocated to the controller processors and short-term also referred to as a local scheduling where the decisions are made regarding how each processor's internal execution policy behaves. This provides the autonomous structure i.e. each processor can have its own different local polices along with one common global scheduling policy. Generally, scheduling in multiprocessor involves Time sharing and Space sharing mechanisms where the former refers to the distribution of various jobs among number of available processors, executing jobs at their intended time intervals whereas the latter allocates the number of processors to a single active job either on the basis of data partitioning or job execution logic thus having the capacity to run multiple jobs in parallel by partitioning them among number of available processors. Space sharing mechanisms require concurrent module embedded logic into the application such that the application can be partitioned based upon their concurrency [11]. Time sharing mechanisms are mostly applicable where number of non-parallelized jobs is encountered and the jobs usually have sequential logic. The scheme covers set of available jobs and a distribution policy & logic according to which jobs are assigned to multiprocessors as shown in the Fig. 1

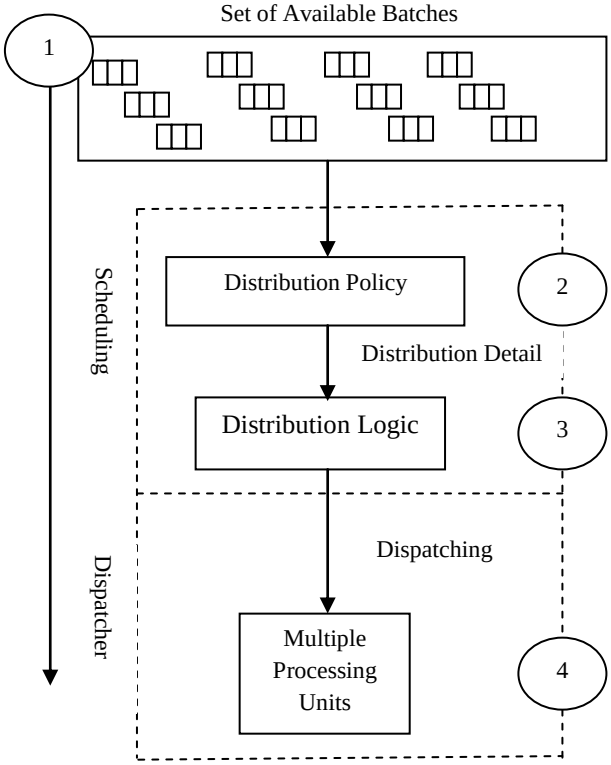

Fig 1: Time-Sharing Scheduling Mechanism

Further the mechanisms can be extended to static and dynamic policies where static scheduling considers only the

guiding principles i.e. job burst_size (No. of CPU burst cycle required), limited processor job buffer queue (ready as well as waiting) and queue order as a distribution parameter. The queue order may be arranged according to the shortest job like in traditional uniprocessor scheduling. Static scheduling policy also states that allocated processors will not relinquish control till a particular job is in execution, whereas dynamic policies measures the present processor's load, maximum percentage of the load that it can handle, their cycle speed etc. Dynamic policies follow the rules of preemption while encountering high priority jobs and consider ongoing job parallelism during different states of execution; this will provide a change in processor allocation as different jobs' execution states may demand different parallelism requirements or parallelism may vary during job execution life cycle [3][4]. Scheduling of jobs among processors and vice versa can be done through the use of multiprocessor scheduling algorithms which are based on scheduling policies such as static and dynamic scheduling. Static scheduling also states that processors are not effectively utilized in case the variability in system load is quite high while dynamic scheduling lies in its ability to adapt itself to changing system conditions. The parameters such as processor load and speed are not calculated in static scheduling algorithms thus making their implementation easier with low scheduling overhead while the calculation of these parameters in dynamic scheduling algorithms makes them complex but highly effective [1].

2. SCHEDULING STRATEGIES

Static scheduling provides few algorithms such as critical long term queue distribution (CLD) with fixed buffer size where number of batches of jobs is either encountered periodically at different time intervals or at same time interval. Now the idea is to distribute these jobs chronologically among number of available processors till the processors have free space in their job buffer queue, as in this approach, the variable processor buffer size has been taken. Whenever the processors' buffer queue gets full, the CLD stops its further distribution until an empty space is created in the buffer queue of any of the available processors and re-continues its distribution logic [5][6]. Each processor individually runs its own round-robin scheduling by calculating time quantum as an average of total no. of computation cycles of its current job queue. Long term queue order can be like first come first serve where batches are allocated as they arrived or queue can be ordered with shortest batch first (SBF) where batches are sorted first i.e. the batch having minimum no. of jobs will be allocated first, note that in this policy only batches are sorted not their jobs. So the batch having smallest job set will be distributed first. Other approach where batches as well as their jobs are sorted and then distributed is referred to as a shortest batch shortest job first (SBSJF). It is the responsibility of the long-term scheduler to sort the batches as well as job queue. Other approach, where each processor can have at most single job for execution without having a processor buffer, is referred to as an effective single job execution policy (ESJE). Each processor's short-term scheduler picks a ready job in front of the long-term queue. No more jobs are allocated until there is an I/O operation or the processor finishes its last job execution. Whenever the processor finishes its current job execution its short-term scheduler immediately assigns it, the next ready job. ESJE long-term queue can be organized as FCFS, SBF, and SBSJF. The benefit from this scheme is that processors are lightly loaded as they have only single job for execution, this has the advantage over CLD where most of the times, the processor buffers are fully loaded. Dynamic scheduling policy requires parameters such as processor load and processor speed to be measured where processor load can be measured as number of processes in the processor's ready queue, also referred to as a processor's job queue length, jobs are scheduled according to the processor having minimum job queue length (MJQL). Here, if the round robin method is applied as local processor scheduling, then there will be two further selection process either the total no. of pending jobs are calculated in the processor's ready queue regardless of the current running job index of the round robin cycle. Other way of selection is to measure the remaining round robin cycle length, which is measured from current running job index to the end of the ready queue. This will provide quicker access to recent incoming jobs and is referred to as QRA (quicker round-robin access). Other policies for homogeneous processors consider remaining processor computation cycles, where jobs are scheduled according to the minimum processor's remaining computation cycles (MPRC).Here, again the remaining computation cycles are computed either by considering total number of pending jobs in the queue or by computing cycles of remaining round robin cycle length. Improvement over MPRC for dynamic distribution involve heterogeneous processors thus care should be taken of the overall workload as jobs are then allocated after measuring processor cycle speed i.e. job allocation to that processor which consumes minimum no. of computation cycles for the completion of that particular job along with its present workload. In other words, the processor which completes the job quickly will take care of that job, also known as cycle based scheduling (CBS). CBS will follow the FCFS as local processor scheduling. During allocation, if there is a tie among processors' speed, then the scheduling decisions can be made by either measuring processor job queue length or remaining computation cycle as described earlier. One another dynamic policy is the one in which the job may demand a particular feature equipped processor (FEP). If found, the job is allocated to that processor otherwise it will have to wait or the job's structure must be moldable to adapt itself according to different characteristic processor. One another solution to this is that if a busy processor is found for the same characteristics, that processor is forced to take care of that particular job also referred to as a feature based scheduling (FBS). Proportionate policies having partial dynamic characteristics involve workload partitioning based on processor speed (PWLP), where the overall workload is proportionally divided in advance in relevance to the processor speed before distribution. In further section, the simulation containing critical job distribution as well as effective single job execution will be discussed. Future versions of this research may consider dynamic policies analysis as well.

3. CONNECTIVITY ARCHITECTURE

General connectivity structure for CLD and ESJE scheduling policies is described as fig-2. Each processor has its own short-term scheduler associated with a pre-fetch buffer. The scheduler selects the job from the long term scheduling queue and places it inside the prefetch buffer. The Dispatcher then picks the job from the prefetch buffers and places it into processors local memory. For CLD if processor buffer has some free memory space to occupy other waiting jobs, the dispatcher immediately allocates other pre-fetched jobs and empties the pre-fetch buffers. For ESJE where each processor has at most single job for execution the dispatcher monitors the processor's working status, if found free, immediately allocates the pre-fetched job to its working memory. So frequency of this scheduler is very high than long term

scheduler. Scheduler requires fast access to one additional job in advance, Some processing units may follows the hyper-threaded architectures where each processor doubles its working space to occupy exactly two job's workload simultaneously along with pre-fetch buffers to schedulers, this hyper threaded approach is necessary to utilize the processor's time stayed idle when the dispatchers picks next ready job from the pre-fetch buffers for further allocation [9].

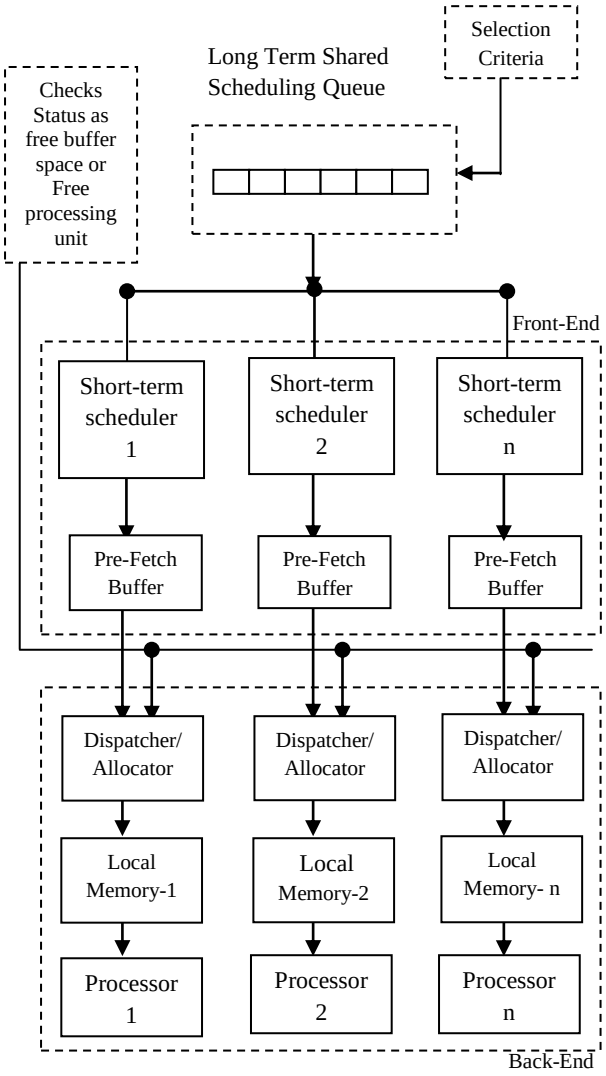

Fig 2: Interconnection Structure

For other dynamic policies the interconnection structure incorporates a load analyzer and a load distributor. Load analyzer measures the dynamic load characteristics of each processor at each cycle, whereas load distributor allocates the jobs according to its associated dynamic criteria. The execution frequency of load analyzer is fast than the load distributor, so before load distribution the current processor load characteristics must reflect to the scheduling decisions [10].

4. SCHEDULING MEASURES

Allocation in MJQL requires minimum queue length i.e. the number of jobs yet not completed in the queue. So, minimum queue length processor can be obtained as-

$$MinQL\_PID = Min\ (\alpha_1, \alpha_2, \alpha_3, \alpha_4 \dots\dots\dots\dots \alpha_n)$$

$$Alloc\ (MinQL\_PID, new\_job)$$

Where α_i represents i^{th} processor's job queue length and $MinQL\_PID$ represents minimum queue length processor id.

Allocation in MRPC requires minimum remaining processor computation time; each job has its associated CPU cycle burst required (No. of CPU cycles)

$$\alpha_i \rightarrow queue\ length\ at\ processor\ i$$

$$CP\_ID \rightarrow Current\ processor\ id$$

$$RP\_CT\ [i] = \sum_{j=0}^{\alpha_i - 1} (rem\_job_j\ \_time)$$

$$CP\_ID = Min\ (RP\_CT)$$

$$Alloc(CP\_ID, new\_job)$$

Cycle based scheduling requires the calculation of processor's cycle speed in terms of no. of cycles per second. Fastest processors may pass many no. of cycles in a second as compare to low speed processors. This approach basically computes the no. of cycles elapsed by processor in one second and then measures the amount of time required to compute current job cycles along with the present workload cycles. The load analyzer firstly perform the summation of pending jobs workload cycles from each processor's ready queue along with the newly arrived job, and then compare it with the processor cycle speed/sec. Processor having smallest no. of timing requirements for the overall workload will take care of newly arrived job. This will perform the balanced load assignment. Consider a processor with 1 GHZ frequency speed.

$$Frequency\ speed - 1GHZ$$

$$Cycles\ Per\ Second - 1000000000$$

$$Sec(S) consumed\ for\ one\ proc.cycle - \frac{1}{1000000000}$$

$$Total\ Time\ to\ Compute\ n - cycles - N * S$$

5. EXPERIMENTAL SETUP

Experimental setup for static scheduling policies (CLD and ESJE) covers logically programmed virtual processors along with their associated job queues. The simulation structure is equipped with integrated synchronized thread oriented environment. Batches of jobs are encountered arbitrarily along with their batch ids. Each batch consists no. of jobs decided arbitrarily during runtime, where workload of each job is associated with a PID and its workload is described in terms of No. of CPU cycles required (burst size). Each virtual processor calculates the time quantum in terms of no. of cycles given to each active job in the queue. Time quantum is calculated periodically by each processor (when there is a change in ready queue) as average of the total no. of burst cycles of jobs in the processor ready queue. Time quantum may vary during execution. The analysis from this research is that as the round robin cycle is increased i.e. total no. of jobs in the round robin scheduling are increased the return cycle of the scheduling is delayed because the incoming jobs are allocated to the virtual processors simultaneously, which increases the length of the queue. Other analysis results describes, as the processor has more buffer space to handle jobs, the time quantum (no. of cycles given to each job) decreases that results in the performance degradation because of the bounded buffer CLD scheduling due to which several context switches may occur. In comparison to CLD scheduling, the ESJE scheduling is more effective because at single point of time ESJE has at most single job for execution.

This will give the advantage that processors are lightly loaded as compare to CLD scheduling, where the processor's buffer are always full/heavily loaded. Other problem with CLD is that the processors requires load balancing issues, predicted results from execution scenarios shows that sometimes some processor's ready queues are fully empty/free and some processors are heavily loaded thus requires some sought of load balancing aspects to be implemented. ESJE gives the advantage because jobs are placed only in the master queue, not allocated to the processor until they are free. Because of this ESJE does not require any load balancing issues. For simulation analysis, varied samples of jobs are taken where no. of jobs are 300, 600, 800 etc. along with their varied no. of required burst cycles.

6. SIMULATED STRUCTURE

Simulated implementation of scheduling policies contain inter-connected thread oriented environment, where each thread corresponds to one virtual processor and a master server thread controls all the child threads and performs synchronization among them. Later the graphical representation of simulation environment will be discussed.

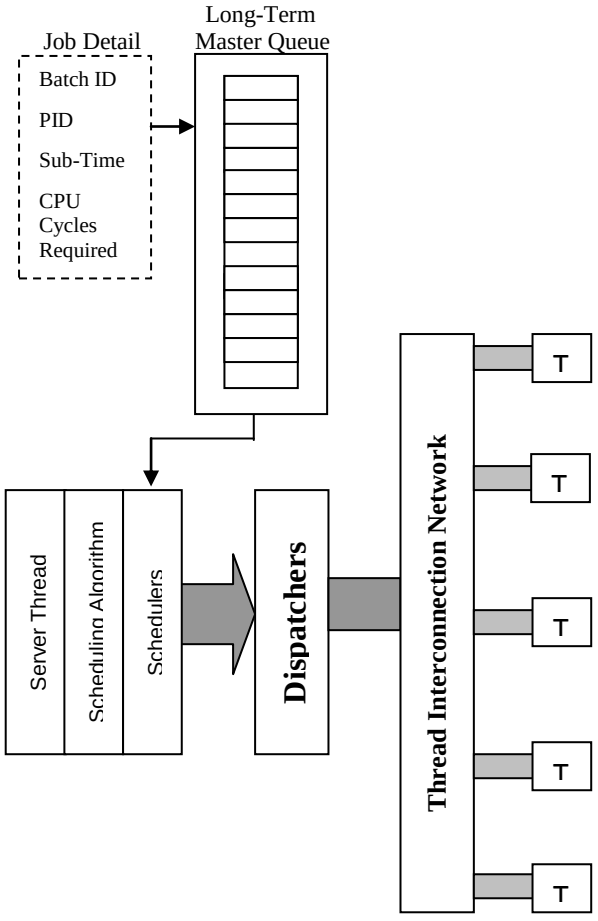

Fig 3: Simulated-Thread Flow Model

Each thread T actually operates like a processing system programmed to process simulated execution flow model as discussed earlier in the connectivity structure. Microsoft thread library is incorporated to perform inter-process communication[7][8]. The problem with this type of structure is that the long term queue defines the critical section i.e. shared by more than one threads in execution, so this will require some sought of thread synchronization or a locking mechanism to avoid simultaneous read access conflicts, as more than one thread may read the same job work space. This behavior of multi-threading synchronization will waste much of the crucial time for job allocation. Future versions may discuss new architectures which will simplify job access via 2-D mesh memory units.

7. EXPERIMENTAL DATA SETS

Data set consists of batch details as well as job details. Each batch has its associated detail like batch arrival time and total no. of jobs encountered in that particular batch. The batch details are dynamically edited as the simulator progresses its execution and updates the batch status as total completed/ uncompleted jobs at one particular time barrier. Burst time is defined as the CPU burst cycles i.e. total no. of CPU cycles required. Data sets are arranged dynamically using arbitrary random number generator. In this research three samples of random data sets are considered under study with total no. of jobs as 300, 600 and 800 respectively.

Batch Detail

Batch_ID	Total_Jobs	Comp. Jobs	Uncomp. jobs

Job Detail

Batch Arr. Time	Batch ID	PID	Sub_Time	Burst Time

As discussed earlier, the problem with bounded buffered scheme is that sometimes some processors are fully free and some of them are heavily loaded. Solution to such inconsistent behavior of multiprocessor requires load balancing algorithms. One proposed solution for load balancing when multiprocessors are in execution is that at each instance of time, load balancing controller monitors the status of each working processor's ready queue and identify the processor which is more heavily loaded and which is very lightly loaded/free, finally, balancing the load among both with allocation of equal no. of computation cycles. Despite of this, one important point regarding load balancing is that it will increase the complexity in system because reassigning jobs from processors queue will waste vital amount of computation time. One another proposed algorithm for load balancing for multiprocessor environment is where each processor's local scheduling is implemented as round robin. Processor gives its intended time quantum to each job in ready queue and moves forward. At the end of round robin cycle the processor moves backward to the start of ready queue. Sometimes the length of round robin queue may increases. Due to this large round robin cycle length, the jobs which are behind to the current round robin process index may wait indefinitely for reallocation of processor again for the start of next round robin cycle. So the idea is to pick up those jobs and perform their allocation to free processors in the environment. Following is the running behavior of simulator which shows the requirement of load balancing, as some of the processors are fully free and some of them are heavily loaded.

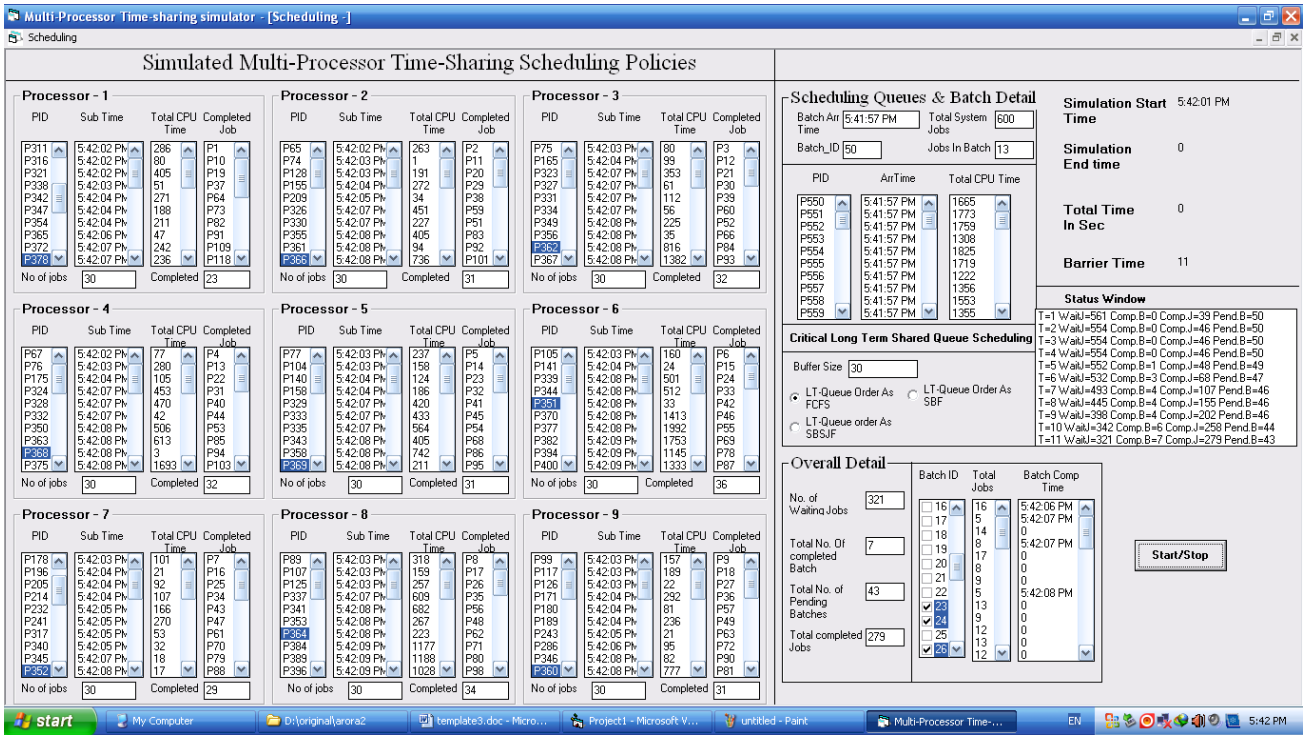

Fig 4: Time Sharing Scheduling Simulator

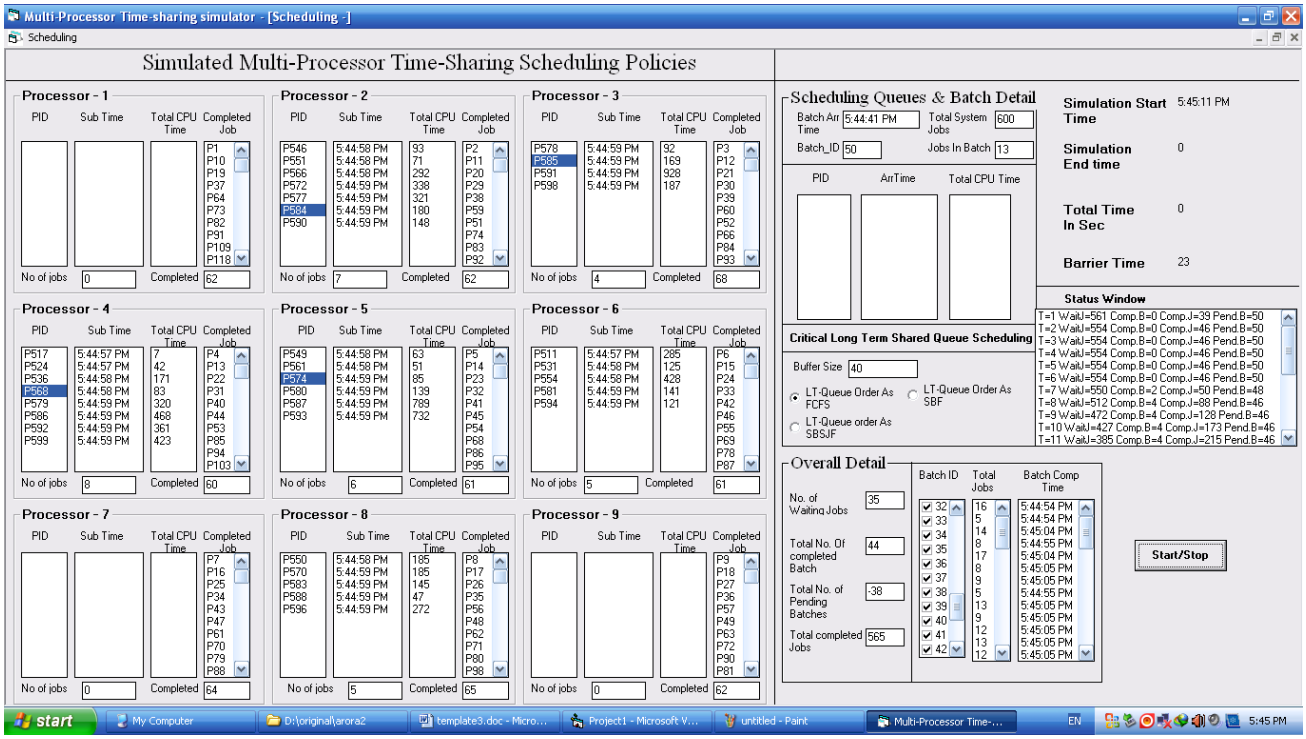

Fig 5: Inconsistent Balanced Load

8. RESULTS & DISCUSSIONS

Following are the illustrations regarding Inferences deduced. The most prominent factors behind such type simulations are the study of system behavior under complex load distributions. In the above scheduling where bounded buffered schemes are employed, the results express that as the processor has longest round robin scheduling queue, the interaction with the system as well as overall time duration increases. Interaction is necessary but overall computation time should not be increased beyond the acceptable threshold time limit. Further the policy SBSJF where firstly batches are sorted and then jobs within the batch are sorted individually as compared to SBF where only batches are sorted i.e. the batch having less no. of jobs is distributed first. The illustrations shown below is captured after distribution of a sample of 600 jobs, conveying message that efforts on sorting individual job order within the batch after sorting batches will not be very beneficial than SBF. Although some differences are noticeable but, very minor because sorting efforts increases as the quantity of jobs increases, this is useless.

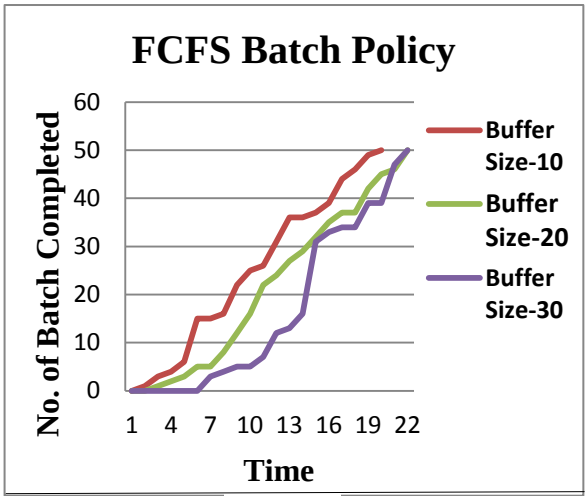

Fig 6: FCFS-Batch Policy

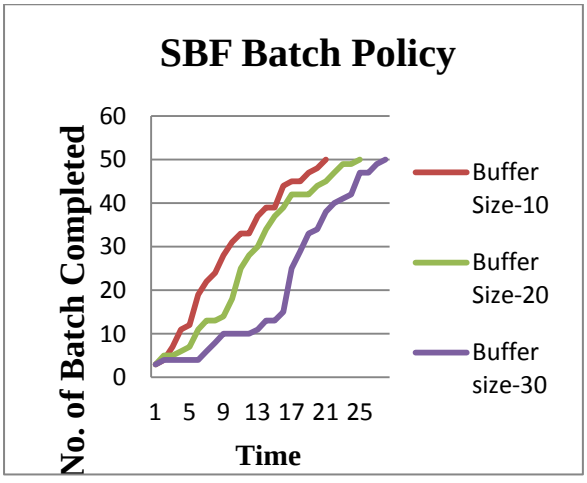

Fig 7: SBF-Batch Policy

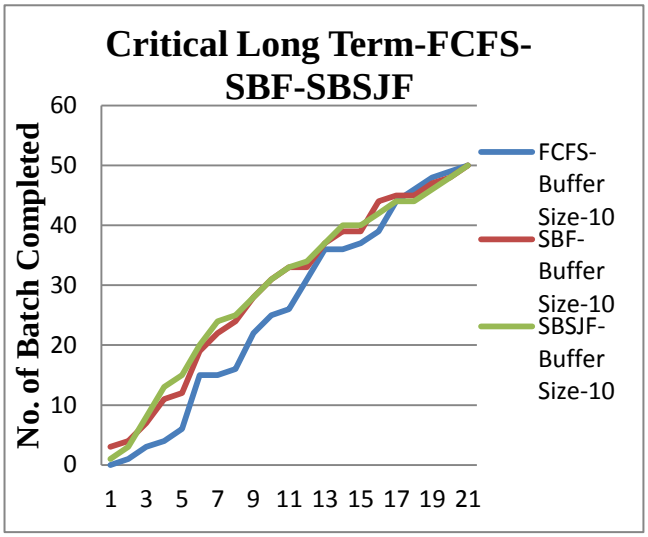

Fig 8: C-Long Term-FCFS-SBF-SBSJF Batch Policy-I

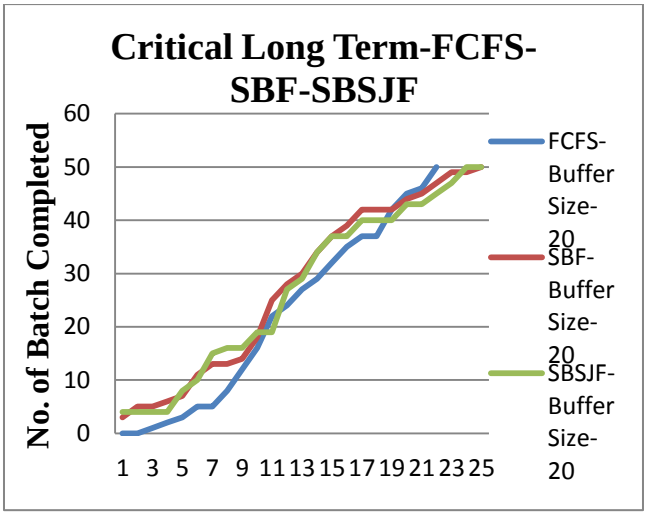

Fig 9: C-Long Term-FCFS-SBF-SBSJF Batch Policy-II

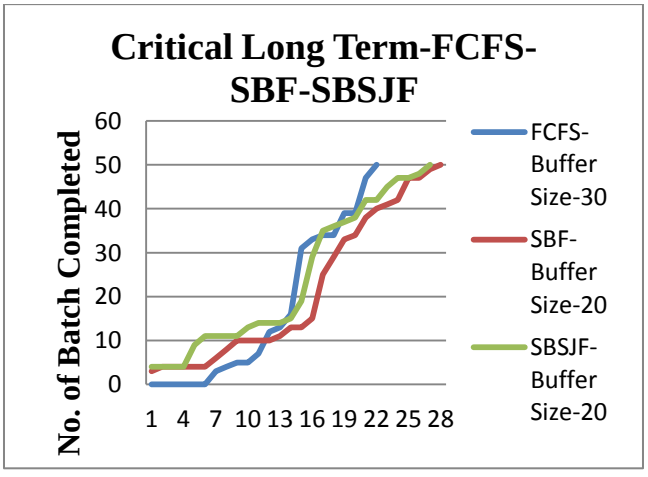

Fig 10: C-Long Term-FCFS-SBF-SBSJF Batch Policy-III

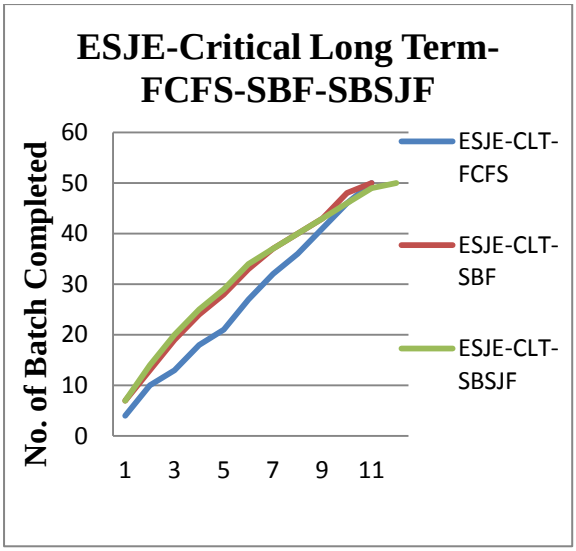

Fig 10: Effective Single Job Execution Policy(ESJE)

9. CONCLUSION AND FUTURE WORK

In this research work from the perspective of Time sharing scheduling, it has been concluded that the system with simultaneous occurred jobs requires much effort during applicability of scheduling/distribution policy. Inaccurate distribution may lead to the inconsistent load allocation, which may further requires reallocation efforts to make effective load balancing. Other points which should be remembered while applying round robin as a local scheduling policy is that as the size of round robin queue increases, its return cycle delays exponentially. In addition the multiple running threads are accessing critical long term queue simultaneously which may lead to concurrent processing around actual multiprocessor implementation. So the job access for multiple processors under uniform memory access will be covered in future research via mesh interconnected memory units. Further study expresses that the dynamic policy implementation will provide more accurate job distribution and avoids load reassignment. In addition giving processor the long buffer for job storage is not very beneficial as discussed in the ESJE policy system. Although interaction with the external environment is minimum but more effective than bounded buffer scheduling schemes.

10. REFERENCES

[1] David, L, Black. 1990. Scheduling and Resource Management Techniques for multiprocessors. Carnegie Mellon University Pittsburgh.

[2] Eric, W. and Kenneth, C. Sevcik 1995 Multiprocessor Scheduling for High Variability Service Time Distributions. University of Toronto.

[3] Nan, G. and Wang, Y. 2012. Fixed-Priority Multiprocessor Scheduling Critical Instant, Response Time and Utilization Bound. IEEE 26th International Parallel and Distributed Processing Symposium Workshops & PhD Forum. Uppsala University Sweden.

[4] Thu, D. Nguyen, Raj, V. & John, Z. 1996 Parallel Application Characterization for Multiprocessor Scheduling Policy Design. Department of Computer Science and Engineering. University of Washington.

[5] Sanjoy, B. Joel, G. 2003 The Static-priority scheduling of periodic task systems upon identical multiprocessor platforms. University of North Carolina at Chapel Hill.

[6] Andersson, B. 2001. Static-Priority Scheduling on Multiprocessors. Real Time System Symposium, 22nd IEEE Conference publication.

[7] Sascha, H. Henri, C. Frederic. S. 2011 From Simulation to Experiment : A Case Study on Multiprocessor Task scheduling. IEEE Symposium on parallel and distributed computing.CNRS/LIG Laboratory, University of Hawai at manoa, Lyon-Villeurbanne, France.

[8] Shivuan, J. Guy, S. Damla, T. 2007 A Performance study of multiprocessor task scheduling algorithms. Springer Science+Business Media, LLC.

[9] Maciej, D. Scheduling Multiprocessor Tasks 1996. European Journal of Operation Research Elsevier

[10] Aryabrata, B. Shelby, F. 2009 An Optimal Scheme for multiprocessor task scheduling- A machine learning approach. University of Georgia USA.

[11] Hsiu-Jy, H. Wei-Ming, L. 2010. Task Scheduling for multiprocessor systems with autonomous performance optimizing control. Journal of information science and engineering. Department of electrical and Computer engineering. University of Texas at san Antonio.