Moldable Load Scheduling Using Demand Adjustable Policies

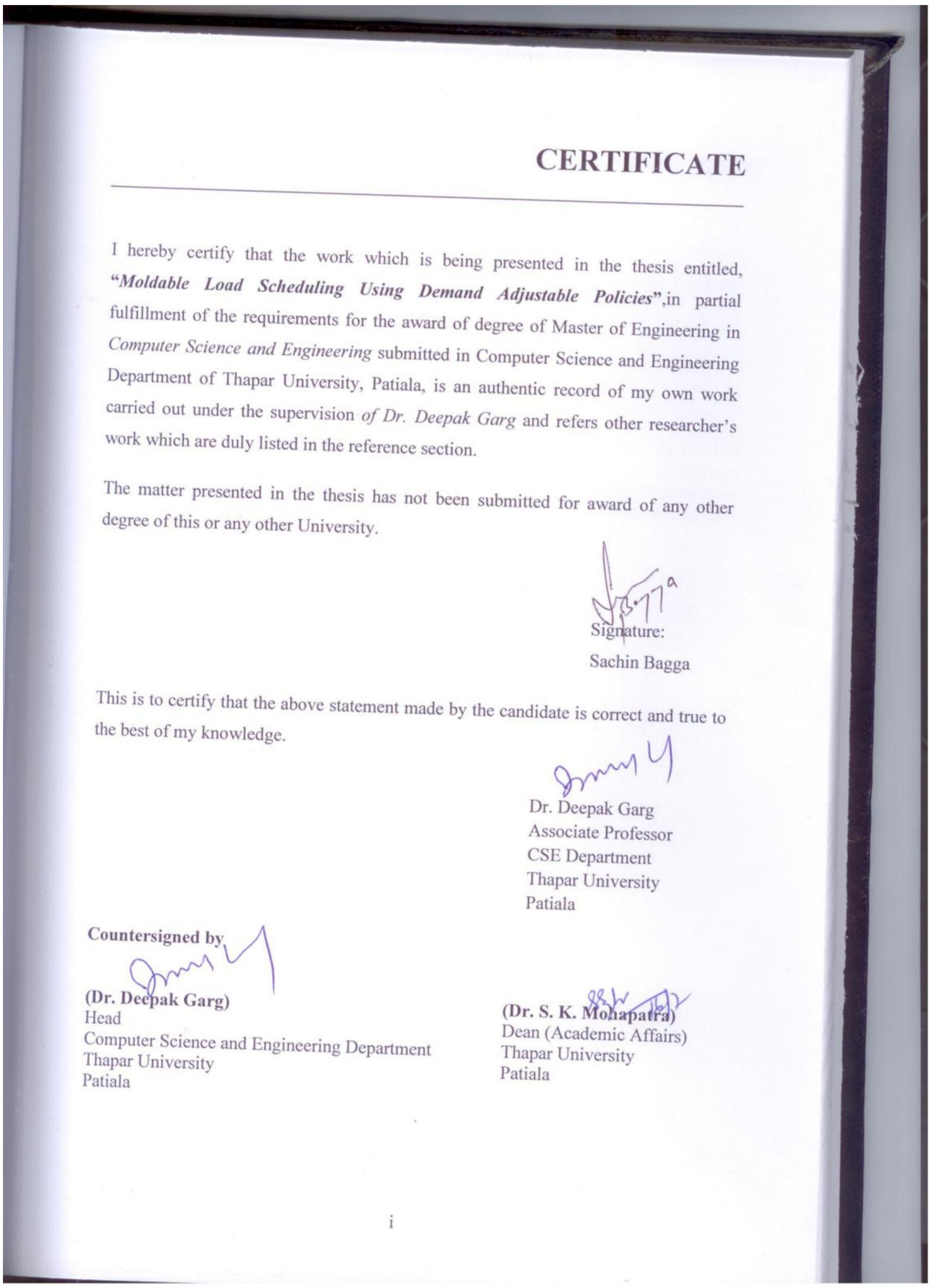
Master of Engineering
in
Computer Science and Engineering

Submitted By
Sachin Bagga
(Roll No.821132009)

Under the supervision of:
Dr. Deepak Garg
Head and Associate Professor

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

June 2014



CERTIFICATE

I hereby certify that the work which is being presented in the thesis entitled, ***"Moldable Load Scheduling Using Demand Adjustable Policies"***,in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Computer Science and Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision *of Dr. Deepak Garg* and refers other researcher's work which are duly listed in the reference section.

The matter presented in the thesis has not been submitted for award of any other degree of this or any other University.

[illegible]

Signature:

Sachin Bagga

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

[illegible]

Dr. Deepak Garg
Associate Professor
CSE Department
Thapar University
Patiala

Countersigned by

[illegible]

(Dr. Deepak Garg)
Head
Computer Science and Engineering Department
Thapar University
Patiala

[illegible]

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

i

Acknowledgment

I express my sincere and deep gratitude to my guide Dr. Deepak Garg, Head and Associate Professor in Computer Science and Engineering Department, Thapar University Patiala, for the invaluable guidance, support and encouragement. He provided me all resource and guidance throughout thesis work.

I would also like to thank to all staff members who were always there at the need of hour and provided with all the help and facilities, which I required for the completion of my thesis .At last but not the least I would like to thank God and mine parents for not letting me down at the time of crisis showing me the silver lining in the dark clouds.

Abstract

Workload distribution among processors is one sided task. Whereas consistent management of processor availability to bulk job arrival is an aspect of resource management. Parallel systems where high probability of infinite job arrivals with varying processor demand requires a lot of adjustment efforts to map processors space over job space .Each job has different required characteristics like no. of processors etc. But the number of available resources is of different characteristics. Particular characteristic processor demanded by a job usually is not available. Such case scenarios are then adjusted to adapt moldable parallel characteristics. Rigid based approaches considered as static demand fit allocation schemes where the job is considered to be active task only when scheduler satisfied the processor demand. Current research focuses on demand adjustment schemes by considering synthetically generated work load and processor availability map with discrete clock frequency. Illustrations produced on the basis of simulation study about demand adjustment schemes consisting static and dynamic approaches with the aim of consistent processor availability fit i.e. processor offered space. Idea behind such experimental study is to analyze various scheduling algorithms along with different performance parameters managing best processor space.

Table of Contents

LIST OF FIGURES

.

List of Tables

Chapter 1
Introduction

1.1 Introduction

Parallel application logic designed on the basis of demand oriented workload structure, where workloads are characterized in terms of number of processor required, decisions for appropriate processing demand will be considered either by number of simultaneous processing threads in a job structure or number of partitioned data sets available. Former specifies control parallel aspects of computation whereas later specifies data parallel aspects of computation. Managing demand based workload in a high data intensive system where infinite job arrival is a very tricky task, because the probability of lesser number of processor available than total demanding needs of the waiting data set is generally high so need of demand adjustable schemes considered as future trend in processor scheduling. Rigid based application structures can be easily executed on systems having bulk of processor offered space (POS) available. Moldable approaches where scheduler takes the decision to map processor available space to job demanding space. The success of such schedulers is dependent upon task adaptability structure (TAS) i.e. capability of task to modify its logical structure to adapt current scenario change. A moldable job can run with any number of processors, within certain range and leaves the decision to the scheduler for parallelism change. The constraint is that only scheduler can decide during runtime the resources can be changed or not but job itself cannot. Resources adjusted at program start, such jobs are moldable. Although, scheduler can manage dynamic change in parallelism at run time (increase or decrease in job demand during execution) despite of this, ongoing parallelism change from jobs perspective is another aspect of task adjustment where tremendous parallelism change in task execution life cycle exists. Jobs which give us the facility of changing the assigned number of resources itself during run time are called malleable jobs. This scheduling is also known a dynamic partitioning. In general parallel jobs provide simultaneous control/data driven modules. In moldable structure where one to one mapping is not possible, how a large scheduling thread will be mapped to minimum possible resource available? The solution to such mapped driven process is to compute computation size of each and every thread of a task and then estimate clock burst of each available processor in one unit time. Now the threads are grouped together corresponding to their collaborate execution cycle and available processor cycle speed. Two or more threads may be given to single individual processor capable to handle intended computation in multitasked concurrent operation, i.e threads are grouped together respective to the processors clock capacity and executed along with context switching.

1.2 Parallel and Distribution System

Parallel computing, as the name suggests is an efficient form of computing **c**apable to execute more than events simultaneously or concurrently with the use of multiple processing units. The processing units may be either a complete CPU (Central Processing Unit) or it can be structured as a multiple ALU's (Arithmetic & Logical Unit) under the control of a common control unit. The idea behind the invention is because of the principal that larger problems can be solved quickly if divided into smaller units and distributed amount to the multiple processing units. Parallel computing programs or algorithms are more difficult to write than sequential ones because now behaviour of the algorithm involves concurrency and this will introduces several new classes of bugs of which race conditions are most common. Parallel computing also not suitable for applications where high amount of dependencies are involved .In such cases the application threads may delay its execution till obtaining the result of its other cooperative threads involved for the production of final outcome .Race condition is nothing but uncertain programming exceptions , which produces unpredictable program state and behaviour due to un-synchronized concurrent events. Race condition originates because of unintentional software coding by programmers .Race condition are hard to find with the conventional debugging methods and tools and requires much of the experience from programmer side. It is an important guideline for the programmer aware of such land mines, which are the strong obstacles into the growth before entering into the dangerous parallel programming zone. The most often race condition are data race conditions, a data race conditions caused due to the simultaneous access on the same memory location.

1.3 Degree of Parallelism

Degree of parallelism can be defined as the maximum event up to which the application can achieve parallelism. No. of levels have been defined and each of which having improved degree of parallelism .Moving one level to another will provide more fine grained parallelism.

i. **Job or Program level:**

   Job level parallelism as the name suggests where multiple programs/process is executed parallel among various processing elements. This is the highest level of parallelism and usually implemented algorithmically.

ii. **Procedure Level:**

   Next highest level of parallelism is the procedure level. Where multiple modules of functions of the application or algorithm will be executed parallel. This type of parallelism will also dependent upon the modular structure of the program.

iii. **Inter - Instruction Level:**

   Next fine grained parallelism level is inter instruction level in this type of parallelism procedures or modules further partitioned among various instructions which are then executed parallel because procedures are self contained block of code which must be written in such a way to achieve parallelism. Data dependency will be more in such types of parallelism .So there might be possible of instruction execution delay.

iv. **Intra - Instruction Level:**

   The last and the highest degree of parallelism is Intra –Instruction level where the instruction is further divided into instruction segments, which are then solved parallel. This type of parallelism requires much of the programming experience and also having high degree of data dependency a compare to previous one. Synchronization, coordination instruction final composition are more difficult.

1.3 Parallel Computing Environments

Various parallel computing environments have been researched based upon various taxonomies:-

i. **Array Processor:**

   Array Processors are the machines having two or more ALU's under the control of a single control unit. These types of systems are capable to perform single instruction multiple data streams. The control unit sends the same instruction to all Processing elements which then performs those similar operations over different parts of the problem.

ii. **Multiprocessor System:**

   Multiprocessor System is the system having two or more Central Processing Units. These are the rue parallel system having the capability to perform parallel computation within a single system. Multi- core processor or chip level processing is now the most common type of parallel computers .Such type of systems are composed of two or more independent core or (CPU's). The cores are typically integrated onto a single integrated circuit die known as a chip multiprocessing or CMP.

iii. **Multi Computer System:**

   Contrasting with multiprocessor, Multi Computers on the other hand are collection of standalone computers connected to form a network. Such systems are also referred to as massively distribute parallel systems. Programming such type of systems are more difficult because synchronization , message passing overheads like sending , accepting, interpreting messages require much of the programming efforts.

1.5 Parallel Job Characterization

1.5.1 Rigid Jobs

Parallel jobs that has the capability to execute on a given separation size are called *rigid jobs.* A real-world job log contains information about scheduling of jobs submitted to a supercomputer over a period of time. For each job scheduled on the system, the information found in a job log generally includes:-

i. The job identification number,
ii. Submission time,
iii. Commence time,
iv. End time,
v. Allocated amount of processors,
vi. Requested maximum time,
vii. Memory needed, Used CPU hours.

All these decisions are taken by the scheduler itself. Such kind of job will not start until the exact requirement of a job is fulfilled, as a result it can lead to the inefficient resource utilization and degradation in the efficiency can occur.

1.5.2 Moldable Job

When the number of resources is adjustable during the execution then the job is called *moldable.* These types of jobs within in specific range can run can run on any number of processor. To evaluate moldable task scheduling policy, in addition to the particulars presented in rigid-scheduling job logs, the following further particulars are required for each job:

i. Choice of processor requirements.
ii. Approximate implementation times matching to the process
iii. Scalability Information.

Number of parallel programming techniques like *PVM, MPI* is there which before starting the execution will ask the user for the number of processors requirements on which the task has to run. Further extending this in *MPI-2*, the user can change the number of resources assigned as per requirement during execution time. Thus MPI-2 supports effectively the concept of malleable jobs which will be discussed in the next section. The moldable jobs can also said to be monotonic if allocation of more resources results in decreasing the execution time and increasing the throughput.

1.5.3 Malleable Jobs

When the jobs give us the flexibility of changing the number of the assigned resources during its execution then such jobs are known as malleable jobs. Such a kind of scheduling is also known as dynamic partitioning**.**

1.6 Developing and Analyzing Parallel System

1.6.1 Cluster Based parallel Programming:-

The cluster based processing can be performed through Sockets, RMI :-

1.6.1.1 Socket Based MPI Communication

End points for communication are defined as Socket. For the purpose of communication in sockets the client server model is used. It is a mixture of a protocol which defines a type of service on the network, the port number, IP address. IP address/Host Name a unique IP address is used for communication. It is of 32 bit specified with four Octets. It is as per the format of IP v4. The advanced communication has provided us the IP address of 128 bits in IP v6 format. Further classification has been done in the form of classes A, B, C, D etc. We can also use a virtual name known as host name for the sake of simplicity. But at the machine level the way in which the communication is done it is with the help of MAC (Media Access Control 48 bit NIC address).

a. **Port Numbers**

   In order to uniquely identify a particular service running on some machine a port number is used. After the connection has been made through which data can be send and receive the port number helps in uniquely identifying it.

b. **Socket Protocol**

   The mode through which communication has to be done like TCP or UDP it is defined through the socket protocol. The reliable connection services are provided by the TCP and unreliable but a very fast communication is provided by the UDP.

The Key Notes during socket programming are as follow:

i. A unique port number and communication protocol must be defined by the server side socket.
ii. Remote host IP, port protocol has to be defined by the socket.
iii. The port number has to be greater than 1024
iv. During connection making process no two or network application should have same port number.

1.6.1.2 RMI Based RPC implementation

i. **Application Layer:-**The actual implementation of client and server takes place at the application layer. An interface that extends java.rmi.Remote must declare number methods. With the help of this interface the client can access these remote methods. The methods which are defined in the one or more interfaces which is extending the java. rmi. Remote interface can be remotely invoked. And at the last the RMI registry is used in order to register the application. Through RMI registry remote object's reference can be get by the client.
ii. **Proxy Layer: -** The stub and skelton at the client and the server respectively are defined at this layer. The remote object's proxy is acted by the stub at the client machine. Similarly for the remote object's proxy at server side is acted by the skelton. Getting the stream of bytes from the Java's byte code is known as Marshalling and it is done by the stub. Similarly the reverse process of getting the byte code is known as Unmarshalling. Due to the stub and the skelton which forms the communication link between the client and remote objects it appears to the client that remote objects is within the JVM(Java Virtual Machine). The rmic compiler creates stub and skelton.
iii. **Remote Reference layer: -** In order to provide the abstraction between the stub and the skelton the remote reference layer is used. A stream oriented connection is made for the data provided by this layer at the transport layer. For the purpose of recovery of lost connection the remote reference layer provides the way out.

iv. **Transport Layer: -** The actual transmission from one machine to another machine in the form of electric signal is done by the transport layer. A stream is created through which sending and receiving of data from one machine to another is accessed by the remote reference layer. Establishing the connection, managing the connection and monitoring are the main tasks of this layer.

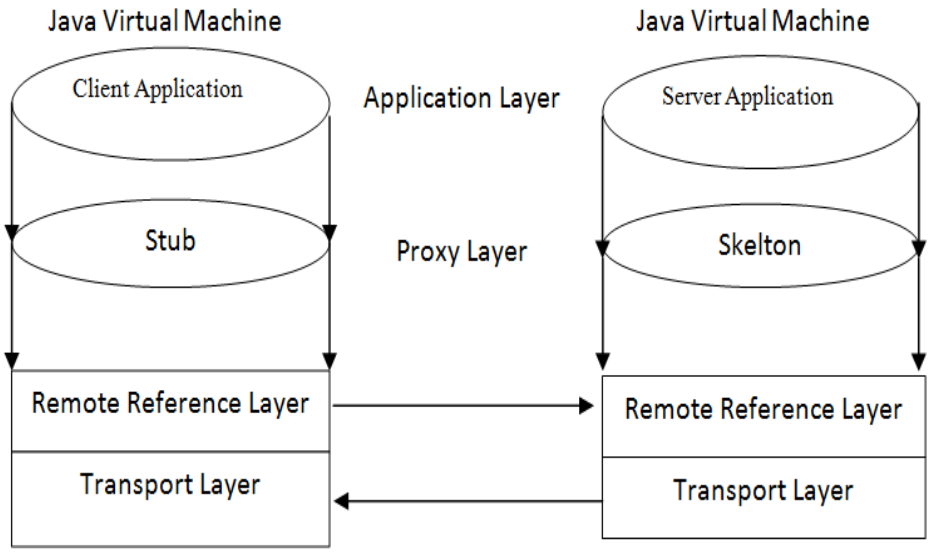


Figure 1.1: Basic architecture of Java RMI

1.6.2 Mathematical Model (Formal specifications)

Formal specification comes under the descriptive design theory using mathematical notations .The purpose of applying formal methods is to perform pre-analysis of software design statements as well as research under observations which may act as benchmark for future implementation. The design statements usually involves mathematical model to elaborate pseudo codes, algebraic specifications, verification, validation design aspects. During development mathematical models are constructed to achieve accurate maturity process. This will provide error free specification analysis in earlier phases of software development. Formal specification methods can be applied in any development segment/phase. Such methods include boolean logic, set theory, qualitative and quantitative variable description. These are written with sound mathematics whose syntax and semantics are formally defined and justified. Such mathematical terms could be successful to represent theoretical aspects in an analytical form. Results could be observed using these mathematical justifications.

Symbolic notations like as-

Actors →activity *Where Activity→Fun1* ∪ *Fun2*…….

∀ x: $P(x)$ means $P(x)$ is true for all x.

∀ $n \in \mathbb{N}$: $n^2 \geq n$

1.6.2 Simulation based experimental studies

Simulation via programmatically designed under some formulated observations of applied research. Such techniques usually come under immediate action research to justify the problem statement via implementation logic. Standard notations are available for asymptotic comparison i.e. standard graphical curves are compared with simulated captured result formulization. After deterministic analysis and justifications immediate action implementation continues to estimate.

Chapter2
Literature Review

2.1 Parallel Computing Models

Traditionally a serial computation involving single core processor is basically a sequential execution. Micro operations are executed one by one along with garbage collection of various CPU registers. This will be required to start immediate next operation occurred in the instruction. So, the delay in instruction execution, results in parallel computation model. Therefore, the various parallel computing models used in simultaneous and concurrent processing are as follows:

Generalized parallel computing model:

- Synchronous PRAM.
- Asynchronous PRAM.

Flynn's Computation Model:

- SISD
- SIMD
- MISD
- MIMD

2.2 Parallel Scheduling

The different number of ways in which we can assign number of processors to various numbers of jobs for simultaneous execution is known as parallel scheduling. There are various manners through which we can implement parallel scheduling, depending upon static policies, dynamic policies, time sharing, space sharing etc. It is discussed in the next section [14].

2.2.1 Types of Scheduling

The scheduling techniques can be broadly specified as:-

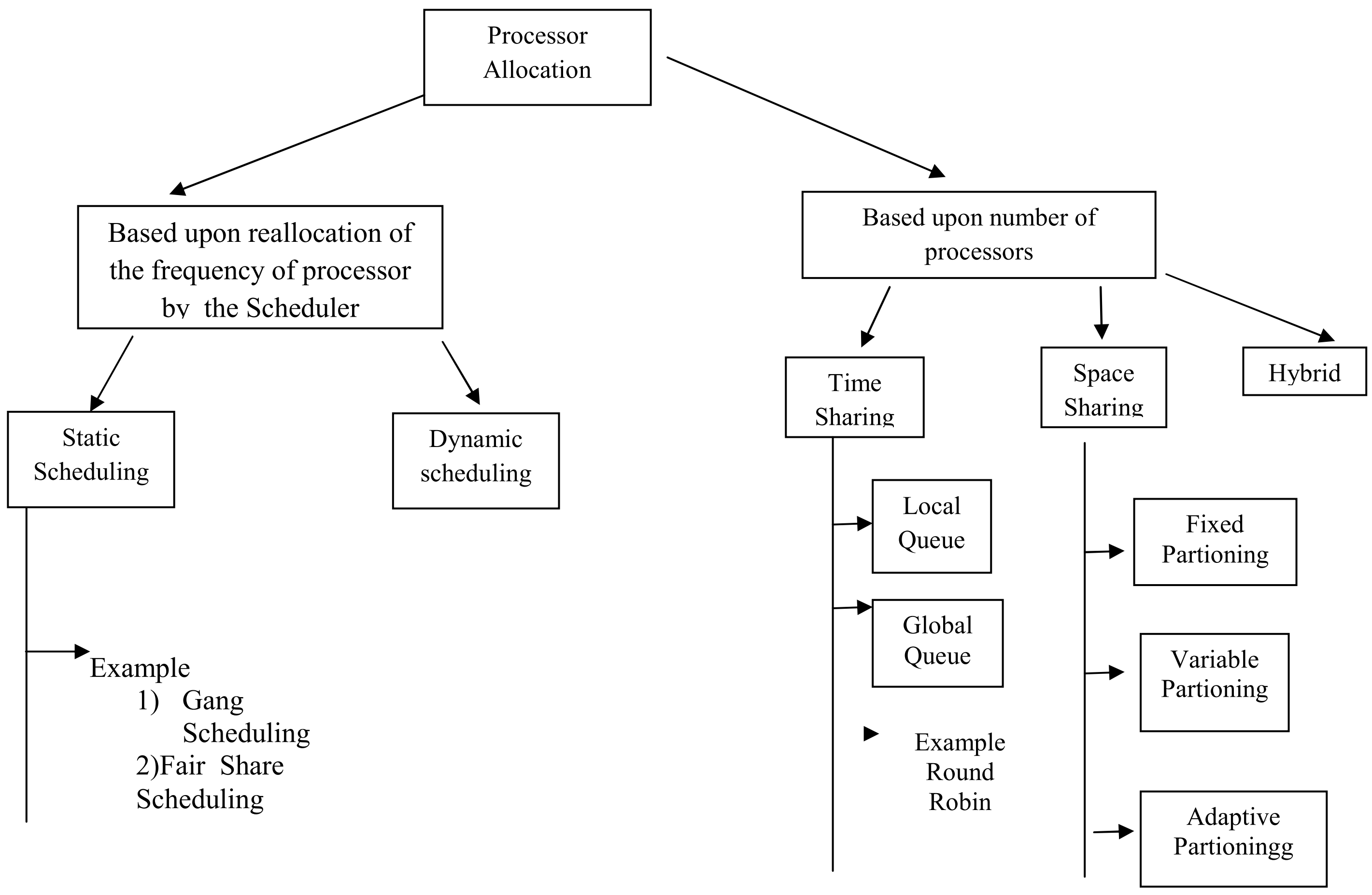


Figure 2.1: Showing ways of scheduling Processors

2.2.2 Static scheduling:

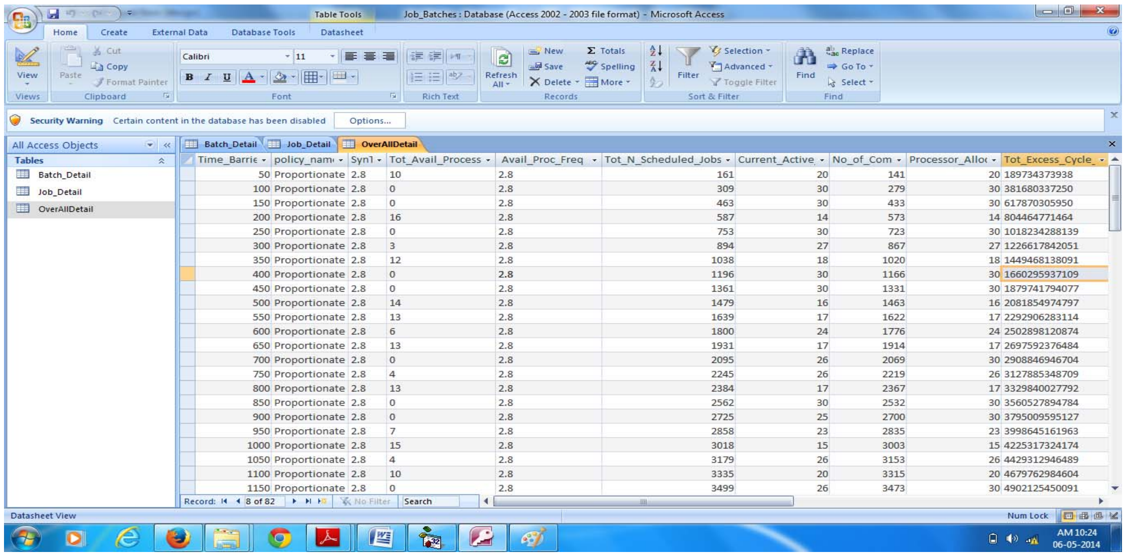
In this type of scheduling once the processor allocation has done, the allocated processors are not reclaimed until the job finishes. This type of allocation can result in ineffective utilization of the system capabilities because of the variations among the system work load .The snapshot of such a database is shown below we can see that each job is having different processor requirements in terms of frequencies, number of processors, CPU burst cycle etc. If for such kind of jobs the static allocation is done then different types of performance factors like excessive cycle length, number of jobs completed per unit time, number of jobs concurrently running per unit time may not be in favour of the user.

2.2.1.1 Fair-share scheduling: In this type of scheduling the usage of the CPU is equally distributed among number of co-ordinating parties. For uniprocessor system number of simultaneously arrived jobs are fairly portioned corresponds to the CPU time computation. In parallel system where number of computation resources is many more along with number of simultaneous arrivals .In such cases computation resources are fairly portioned among coordinating units. Suppose a CPU having 1000 cycles in one unit time which has to distribute among 5 jobs where each job has varied numbers of threads as described:

Table 2.1: Showing Distribution of CPU cycles according to fair share policy

CPU Burst Cycles 1000											
Process 1 **200**					**Process 2** **200**			**Process 3** **200**		**Process 4** **200**	**Process 5** **200**
TU1	TU2	TU3	TU4	TU5	TU1	TU2	TU3	TU1	TU2	TU1	TU2
40	40	40	40	40	66.6	66.6	66.6	100	100	200	200

2.2.1.2 Gang Scheduling: - In this type of scheduling the scheduler schedules the number of threads or processes of the same nature in such a way that the same type of processes execute at the same time on different number of processors. The threads of the same nature that are running simultaneously shall belong to the same process or they might belong to different processes.



Time_Barrie	policy_nam	Syn1	Tot_Avail_Process	Avail_Proc_Freq	Tot_N_Scheduled_Jobs	Current_Active	No_of_Com	Processor_Alloc	Tot_Excess_Cycle_
50	Proportionate	2.8	10	2.8	161	20	141	20	189734373938
100	Proportionate	2.8	0	2.8	309	30	279	30	381680337250
150	Proportionate	2.8	0	2.8	463	30	433	30	617870305950
200	Proportionate	2.8	16	2.8	587	14	573	14	804464771464
250	Proportionate	2.8	0	2.8	753	30	723	30	1018234288139
300	Proportionate	2.8	3	2.8	894	27	867	27	1226617842051
350	Proportionate	2.8	12	2.8	1038	18	1020	18	1449468138091
400	Proportionate	2.8	0	2.8	1196	30	1166	30	1660295937109
450	Proportionate	2.8	0	2.8	1361	30	1331	30	1879741794077
500	Proportionate	2.8	14	2.8	1479	16	1463	16	2081854974797
550	Proportionate	2.8	13	2.8	1639	17	1622	17	2292906283114
600	Proportionate	2.8	6	2.8	1800	24	1776	24	2502898120874
650	Proportionate	2.8	13	2.8	1931	17	1914	17	2697592376484
700	Proportionate	2.8	0	2.8	2095	26	2069	30	2908846946704
750	Proportionate	2.8	4	2.8	2245	26	2219	26	3127885348709
800	Proportionate	2.8	13	2.8	2384	17	2367	17	3329840027792
850	Proportionate	2.8	0	2.8	2562	30	2532	30	3560527894784
900	Proportionate	2.8	0	2.8	2725	25	2700	30	3795009595127
950	Proportionate	2.8	7	2.8	2858	23	2835	23	3998645161963
1000	Proportionate	2.8	15	2.8	3018	15	3003	15	4225317324174
1050	Proportionate	2.8	4	2.8	3179	26	3153	26	4429312946489
1100	Proportionate	2.8	10	2.8	3335	20	3315	20	4679762984604
1150	Proportionate	2.8	0	2.8	3499	26	3473	30	4902125450091

Figure 2.2: Showing number of jobs of same nature active concurrently

2.2.2 Dynamic Scheduling :-In these policies the reallocation of the assigned processors to a particular workload can be done as per the requirement .For reallocation to be done the jobs in the given work load should be in such a way that they can run on different number, types of processors, than the initial requirements of the job. In dynamic scheduling we can change the number of processors requirements before the execution begins, during the execution. The dynamic scheduling gives better results. Assume initially before execution of the job if a job requires C_1 clock cycles and B_1 burst cycle then after t_1 time the requirements will not be same after a certain time t_2 and we can reclaim the resources initially assigned.

2.3 Research towards Workload Characterization

2.3.1 Work Load Characterization:

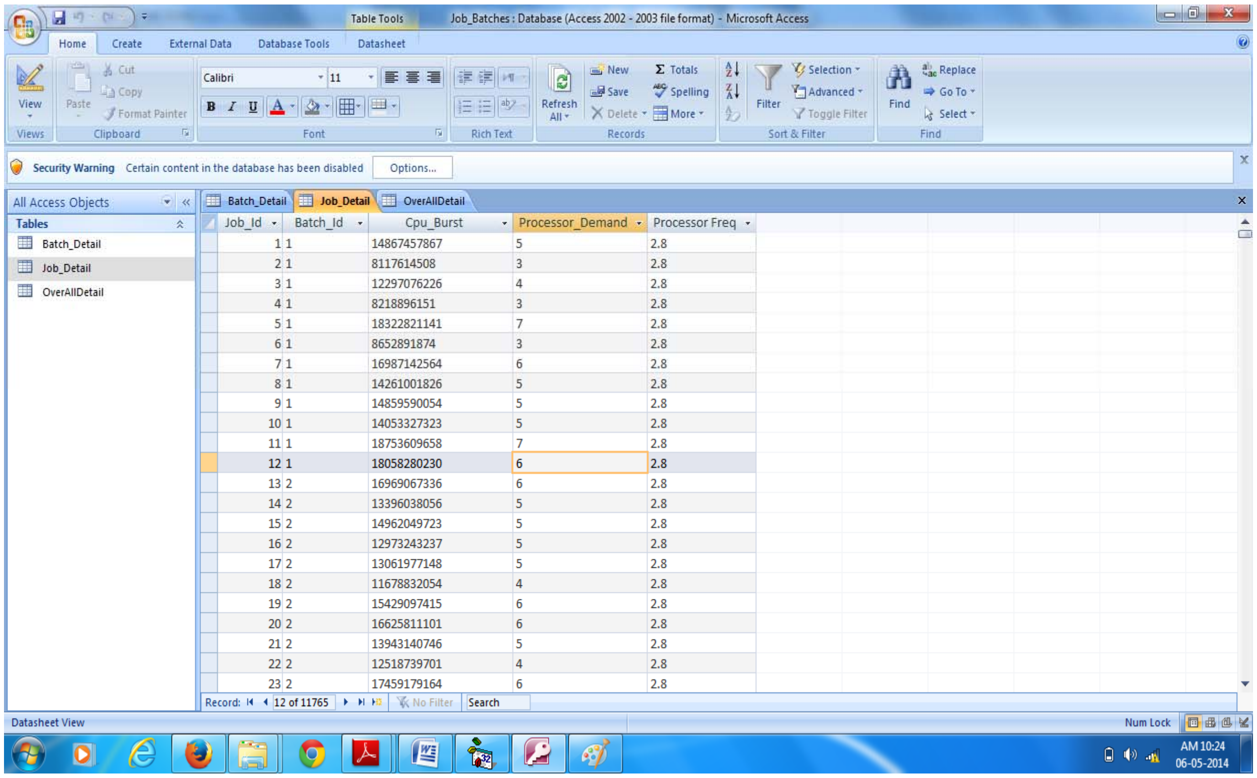
It is classified into types:

- Synthetic workload
- Actual workload

2.3.1.1Synthetic Work Load

Sample based simulation experiment is performed where synthetic workload is generated using random distribution. Simultaneous job arrival may exist, so batches of jobs are organized. Each batch contains number of jobs arrived at same instance with

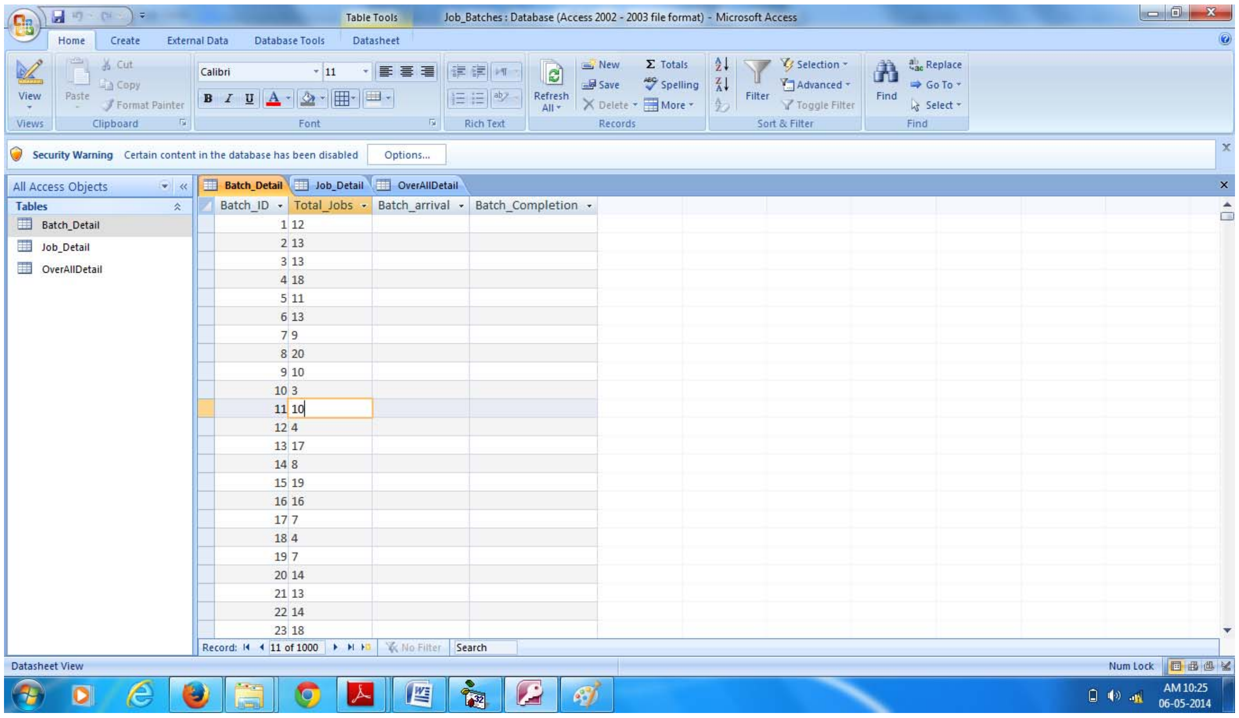
characteristics like *Job Id, CPU Burst*, *Processing Demand*. Overall simulation structure consists of *processor availability space also known as POS (processor offered space), their frequency clock*, *Front End Job Queue* along with *Batch Id* and their respective *no. of jobs*, overall *allocation status, currently active job status, simulation start and end time, Excessive cycle length, Policy Detail etc.*. In general simulation design is based upon multithreading synchronization. Each thread is intended to perform their assigned work like maintaining incoming batch queue, overall allocation status, organizing currently active scheduling list, lists of completed jobs etc. As described each job having its unique identification (JID) and burst cycle refers to the estimated time in terms of CPU cycle required. Processing demand specifies range of processor required by each job, Synthetic work load front end queue (SWFEQ) is generated according to specific processor configuration criteria, during schedule synthetic load will be mapped to current processor configuration available irrespective to the configuration specified during load generation. ECL (Excessive cycle length) specifies extra cycles (exhausted) of allocated processors in terms of each job execution. During job life cycle there may be a situation arise where the processor allocated earlier is much more than required, this is because as the job move towards their final stage of completion the parallelism may change. In this case excessive cycle length will be computed. Although scheduler may obtain excessive processors from such jobs and allocated them to next waiting jobs in the front end job queue. This will provides the benefits that more no. of currently active list. But readjustment during job completion end may be more costly because execution paused and then restarted after demand adjustment. Otherwise if not obtained excessive processors from within job's execution life cycle, the ECL value may be high on each time barrier. Up to this time this is the hypothetic view, further these scenarios will be analyzed along with different scheduling schemes by above described parametric aspects to decide which scheduling scheme is best suited on which situation.
Following are the simulated workload generation for various jobs:


Job_Batches : Database (Access 2002 - 2003 file format) - Microsoft Access
Security Warning Certain content in the database has been disabled
Batch_Detail
Job_Detail
OverAllDetail

Job_Id	Batch_Id	Cpu_Burst	Processor_Demand	Processor Freq
1	1	14867457867	5	2.8
2	1	8117614508	3	2.8
3	1	12297076226	4	2.8
4	1	8218896151	3	2.8
5	1	18322821141	7	2.8
6	1	8652891874	3	2.8
7	1	16987142564	6	2.8
8	1	14261001826	5	2.8
9	1	14859590054	5	2.8
10	1	14053327323	5	2.8
11	1	18753609658	7	2.8
12	1	18058280230	6	2.8
13	2	16969067336	6	2.8
14	2	13396038056	5	2.8
15	2	14962049723	5	2.8
16	2	12973243237	5	2.8
17	2	13061977148	5	2.8
18	2	11678832054	4	2.8
19	2	15429097415	6	2.8
20	2	16625811101	6	2.8
21	2	13943140746	5	2.8
22	2	12518739701	4	2.8
23	2	17459179164	6	2.8

Record: 12 of 11765

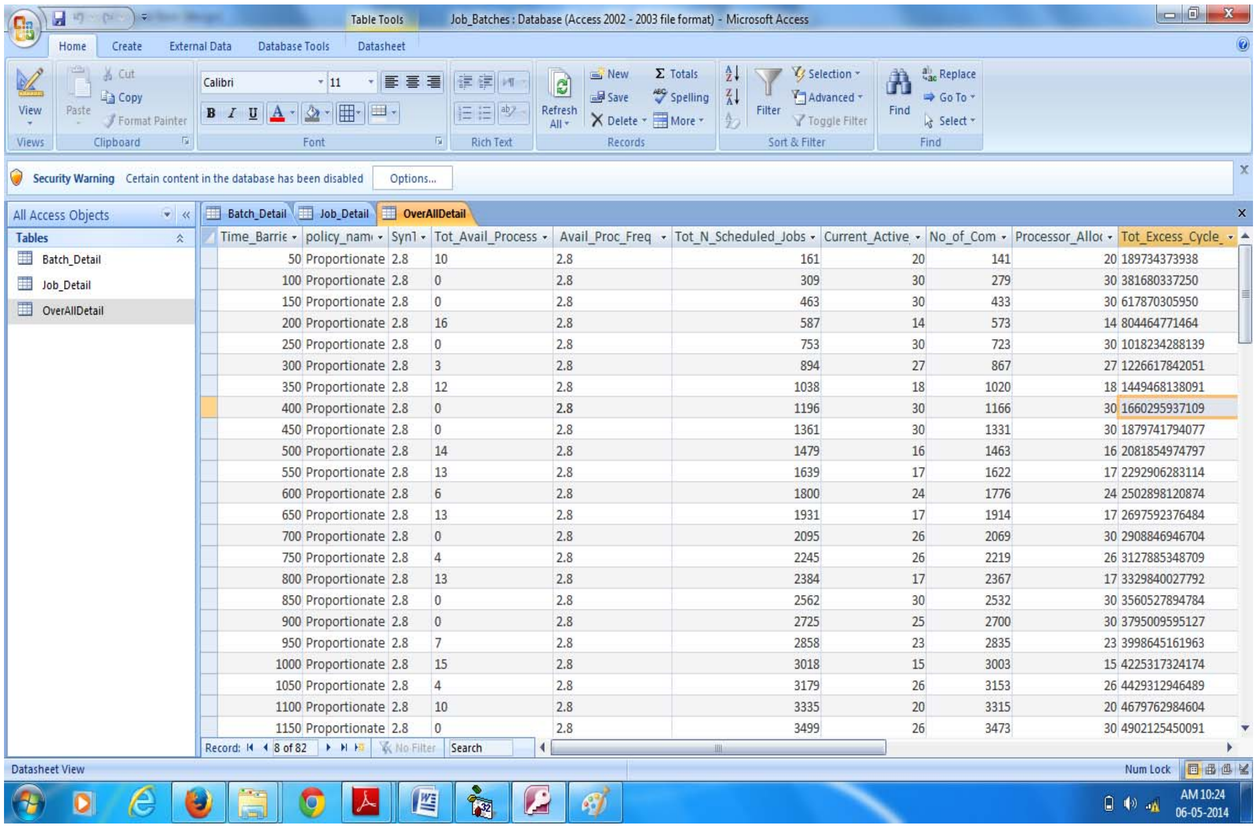
Figure 2.3: Showing various Jobs characteristics


Job_Batches : Database (Access 2002 - 2003 file format) - Microsoft Access
Security Warning Certain content in the database has been disabled
Batch_Detail
Job_Detail
OverAllDetail

Batch_ID	Total_Jobs	Batch_arrival	Batch_Completion
1	12		
2	13		
3	13		
4	18		
5	11		
6	13		
7	9		
8	20		
9	10		
10	3		
11	10		
12	4		
13	17		
14	8		
15	19		
16	16		
17	7		
18	4		
19	7		
20	14		
21	13		
22	14		
23	18		

Record: 11 of 1000

Figure 2.4: Number of jobs arrived simultaneously in a single Batch



Job_Batches : Database (Access 2002 - 2003 file format) - Microsoft Access

Security Warning Certain content in the database has been disabled

Time_Barrie	policy_nam	Syn1	Tot_Avail_Process	Avail_Proc_Freq	Tot_N_Scheduled_Jobs	Current_Active	No_of_Com	Processor_Alloc	Tot_Excess_Cycle_
50	Proportionate	2.8	10	2.8	161	20	141	20	189734373938
100	Proportionate	2.8	0	2.8	309	30	279	30	381680337250
150	Proportionate	2.8	0	2.8	463	30	433	30	617870305950
200	Proportionate	2.8	16	2.8	587	14	573	14	804464771464
250	Proportionate	2.8	0	2.8	753	30	723	30	1018234288139
300	Proportionate	2.8	3	2.8	894	27	867	27	1226617842051
350	Proportionate	2.8	12	2.8	1038	18	1020	18	1449468138091
400	Proportionate	2.8	0	2.8	1196	30	1166	30	1660295937109
450	Proportionate	2.8	0	2.8	1361	30	1331	30	1879741794077
500	Proportionate	2.8	14	2.8	1479	16	1463	16	2081854974797
550	Proportionate	2.8	13	2.8	1639	17	1622	17	2292906283114
600	Proportionate	2.8	6	2.8	1800	24	1776	24	2502898120874
650	Proportionate	2.8	13	2.8	1931	17	1914	17	2697592376484
700	Proportionate	2.8	0	2.8	2095	26	2069	30	2908846946704
750	Proportionate	2.8	4	2.8	2245	26	2219	26	3127885348709
800	Proportionate	2.8	13	2.8	2384	17	2367	17	3329840027792
850	Proportionate	2.8	0	2.8	2562	30	2532	30	3560527894784
900	Proportionate	2.8	0	2.8	2725	25	2700	30	3795009595127
950	Proportionate	2.8	7	2.8	2858	23	2835	23	3998645161963
1000	Proportionate	2.8	15	2.8	3018	15	3003	15	4225317324174
1050	Proportionate	2.8	4	2.8	3179	26	3153	26	4429312946489
1100	Proportionate	2.8	10	2.8	3335	20	3315	20	4679762984604
1150	Proportionate	2.8	0	2.8	3499	26	3473	30	4902125450091

Record: 8 of 82

Figure 2.5: Snapshot of a data produced during execution of jobs

2.3.1.2 Actual Work Load

Is a real time job which is running in the processor's memory takes input stimuli and Output stimuli? Such jobs utilizes CPU time, register for its operational task. Such tasks have real time interaction along with system components and the user. Basically a computation process in real code process consists of control flows basis on the conditional expression, iterative flows, program sequence control in addition code regeneration by compiler for performance efficiency.

2.3.2 Characterizing Moldable Parallel Jobs

Walfredo Cirne and Francine Berman (2001) outlined that the type of input given to the supercomputer scheduler effects a lot on its performance. So it is important that before evaluation of super computer scheduler the workload must be effectively reviewed. The rigid parallel jobs require that they must be partitioned into fixed sizes in order to run effectively. The moldable jobs which have the capability to run on a different number of partitioned size have majority in certain kinds of jobs called parallel jobs. In this paper by using good analytical models and based upon user survey a workload model for moldable jobs is described. In order to develop a performance efficient strategies for the selection of job partition size and for the enhancement of supercomputer scheduler the model proposed by him can be directly applied.

Allen B. Downey in (1997) by observing a large number of parallel computers in the Cornell theory centre and San Diego super computer centre developed a workload model. This model helps us in checking the performance of various strategies while scheduling the moldable jobs on a parallel systems having space sharing architecture. In his research they reach to the conclusion that Adaptive static partitioning (ASP) which was supposed to work in a effective manner for other workloads, is not performing very well compared to the strategies that adapt the system load. The best strategy he considered one is that helps in reducing allocations when high amount of load is there. [12]

J.T. Moscicki, M. Lamannaa, and M. Bubak (2012) shows that performance and reliability of large grid infrastructures may suffer from large and unpredictable variations. In this paper the impact of the job queuing time on processing of moldable tasks which are commonly found in large-scale production grids has been studied. They use the mean value and variance of make span as the quality of service indicators. The general task processing model which provides a quantitative comparison between two models: early and late job binding in a user-level overlay applied to the EGEE Grid infrastructure has been developed. In this research, they find that the late-binding model effectively defines a transformation of the distribution of makespan according to the Central Limit Theorem. As demonstrated by Monte Carlo simulations using real job traces, this transformation allows to substantially reducing the mean value and variance of makespan. For certain classes of applications task granularity may be adjusted such that a speedup of an order of magnitude or more may be achieved. He use this result to propose a general strategy for managing access to resources and optimization of workload based on Ganga and DIANE user-level overlay tools. Key features of this approach include: a late-binding scheduler, an ability to interface to a wide range of distributed systems, an ability to extend and customize the system to cover application-specific scheduling and processing patterns and finally, ease of use and lightweight deployment in the user space. They discusses the impact of this approach for some practical applications where efficient processing of many tasks is required to solve scientific problems.[6]

2.4 Effective Scheduler Characteristics

They Salient Features of a effective scheduler are as follow:

i. **Dynamic**:-The scheduler must have capability to process the load changing in processors as well as demand changing in execution of the given job set and should be capable of providing the given amount resources in an effective manner.

ii. **Effective resource Mapping** – Effectiveness in terms of Resource mapping i.e. effective mapping provides increased throughput as well as reduced task adjustment efforts.

iii. **Synchronized Thread**- scheduler must ensure the synchronization of running threads in network based communication flows, although simulation may also require synchronization aspects but minimum as compare to network based parallel designs.

iv. **Transparency: -** The transparency is in the terms of execution of the task either from the local or remote. Same set of the results must be produced from local and remote machine. The user must have ease of such a way that whether remote execution is going on or local execution is going on. For developing such types of facilities certain programming expects like RMI(Remote method invocation) in JAVA is very helpful, where it seems to a user that local calling of a function is going on but a function which has been called is actually existing on some another machine.

v. **Fairness**: - It is concerned with the aspect that each demand must be fulfilled in a best affective manner. So that the given amount of resources are effectively distributed among various requirements. Further it is also as per the user requirement that a thread level or process level fairness has to be provide. Depending upon whether to schedule large number of jobs or earlier completion of lesser number of jobs is required.

vi. **General purpose**: - As different set of load can arrive comprising of different set of applications like some can be real time jobs requiring space sharing scheduling, non iterative batch jobs which might require time sharing scheduling. So scheduler must provide up to some extent of the services for any type of job arriving in general.

Chapter3
Proposed Work

3.1 Problem Formulation

Parallel distribution of workload among the number of the processors is not only the task through which we can achieve high performance but also regular management of allocation of the processors to the large number of the job arrival is of also large importance .In super computer systems there are sort of infinite job arrivals with varying amount of requirements from system like number of processors, varying frequency of processors requires, number of jobs to be run in a parallel, sequential way etc that is the reason that a lot of adjustment effort is required to map processors space over job space. Main problem arises because such required characteristics for each job is not available and an efficient amount of dynamic parallel molding for requirements of jobs has to be done in order to achieve high performance in terms of number of simultaneous execution of jobs, more execution of the threads of a single job at a given time. Certain types of jobs called moldable jobs can provide us the facilities such that we can change the requirements of the job but these decisions has to be taken before the starting of the execution of the job. The effectiveness will depend upon task adaptability structure and the way the scheduler is assigning the resources. In the thesis three dynamic polices has been discussed through which the effective resource management can be done in a better way.

The main objectives of the thesis are:-

i. Distribution of jobs with efficient resource mapping.
ii. Managing throughput in terms of simultaneous thread/process execution.
iii. Achieving demand adjustment benefit by overlapping processor space to job space.
iv. Adjusting clock speed variations before actual demand adjustment needs.
v. Considering effect of dynamic parallelism change during job execution life cycle.

3.2 Space sharing policy

Space sharing policy where each job may have more than one processor, single job having multiple thread in action, distributed among two different processors for simultaneous execution. This is necessary because sometimes multiple threads perform inter-process communication, for achieving parallelism job scheduling requires multiple processors in execution.

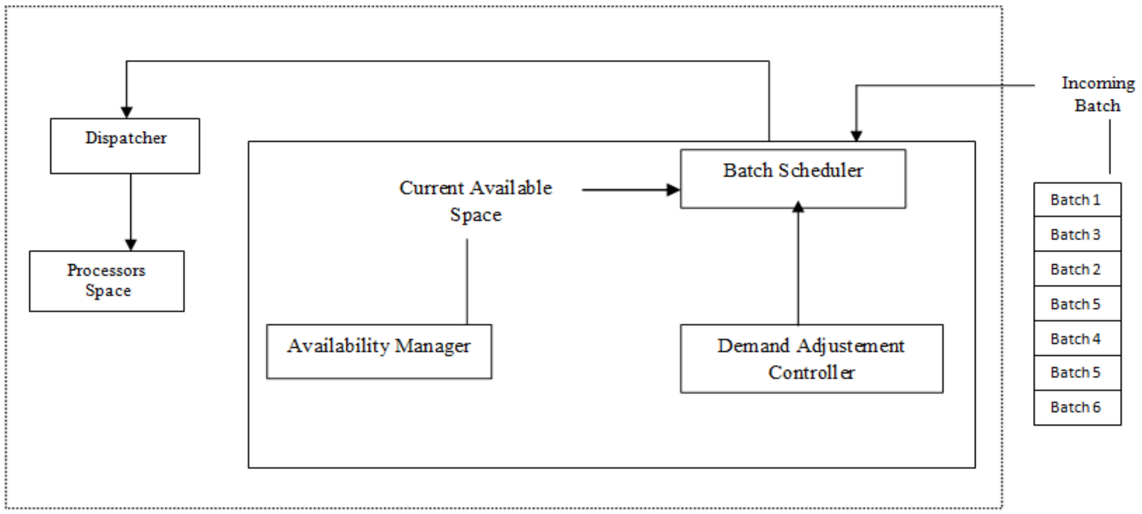


Figure 3.1: Showing space sharing policy in action

3.3 Demand Based Model for Moldable Scheduling

Parallel job workload involves multiple job arrivals consisting varying processing demand, sometimes not fulfilled due to the current availability requires, adjustment as described by below process model. Generally moldable or malleable structures are used in schedulers, because in both cases, the demand adjustment is required. In moldable, the scheduler performs the demand adjustment according to the current availability and no. of tasks yet not allocated. The decision is in the hand of schedulers, performs resource management. In malleable scheduling, the decision of demand adjustment is dynamically performed but on the request of jobs, this decision is taken by job itself, although adjustment is performed by scheduler or its intended component but after issuing the request by job, demand may be decreased or increased as job request, schedulers component will manage availability and demand requests, such jobs are malleable, job controller has the functionality to manage processing demand for each of its running thread as required. Rigid jobs are the fixed will not active in execution until demand request fulfilled.

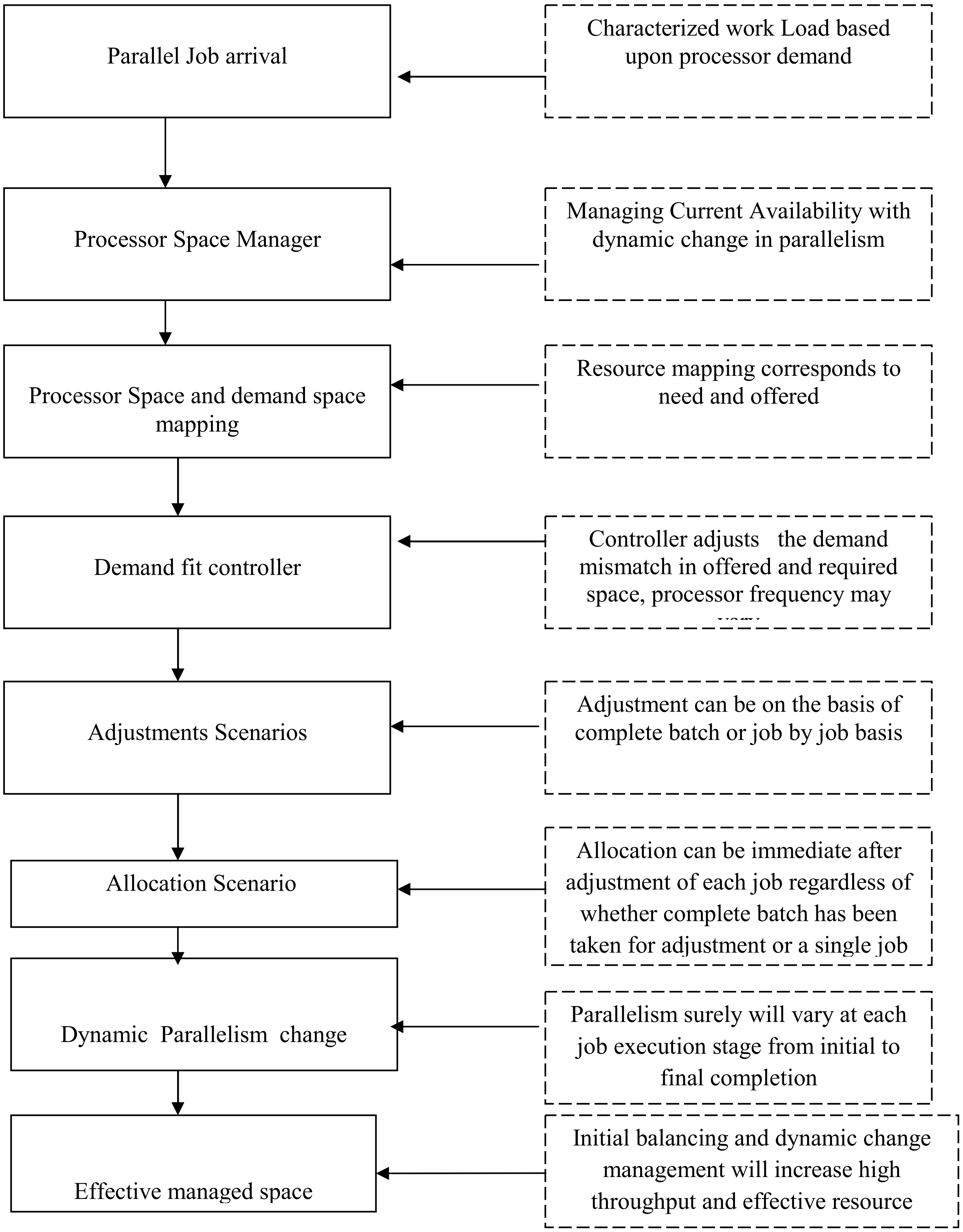

Parallel Job arrival
Characterized work Load based upon processor demand
Processor Space Manager
Managing Current Availability with dynamic change in parallelism
Processor Space and demand space mapping
Resource mapping corresponds to need and offered
Demand fit controller
Controller adjusts the demand mismatch in offered and required space, processor frequency may
Adjustments Scenarios
Adjustment can be on the basis of complete batch or job by job basis
Allocation Scenario
Allocation can be immediate after adjustment of each job regardless of whether complete batch has been taken for adjustment or a single job
Dynamic Parallelism change
Parallelism surely will vary at each job execution stage from initial to final completion
Effective managed space
Initial balancing and dynamic change management will increase high throughput and effective resource

Figure 3.2: Showing Demand Based Model for Moldable Scheduling

3.4 Approaches to Demand Based Allocation Schemes

Several approaches have been developed, some of them are developed on the basis of rigid based job characterization, other are on the bases of moldable and malleable demand adjustment approaches. Selection of a particular approach will depend upon the current availability and requirement currently ready for schedule. Most of the times adjustment is required.

3.4.1 Strictly Demand Fit Allocation (SDF)

Scheduling decisions where processor demand for each job is fixed and allocated with as much as processor required is a kind of strictly demand fit allocation (SDF). Purely a static processor space division where a job is not executed until the required number of processors are not available.

```
For each JID in FrontEndJobQueue
do
    $Set\ Curr\_Dem \leftarrow Get\_Pr\_Demand(JID)$
    $Set\ Curr\_Avail \leftarrow Get\_AvailPr(\ )$
    If   Curr_Dem <= Curr_Avail   then
        $SetMode(Active, JID)$
        $Allocate(JID, Curr\_Dem)$
        $Set\ Curr\_Avail \leftarrow\ Curr\_Avail -\ Curr\_Dem$
        Set_AvailPr(Curr_Avail)
    Else
    $SetMode(Wait, JID)$
    endif
endfor
```

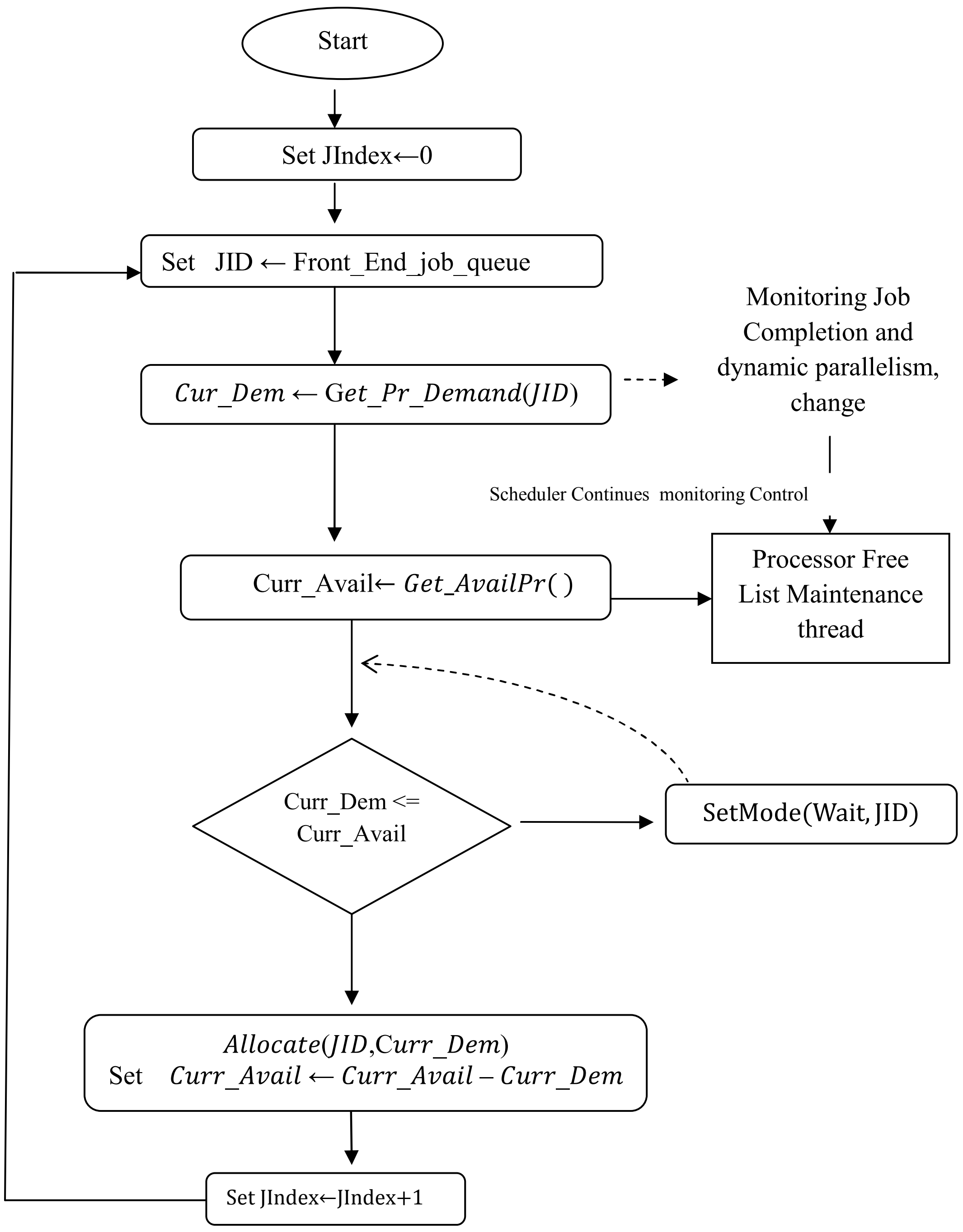
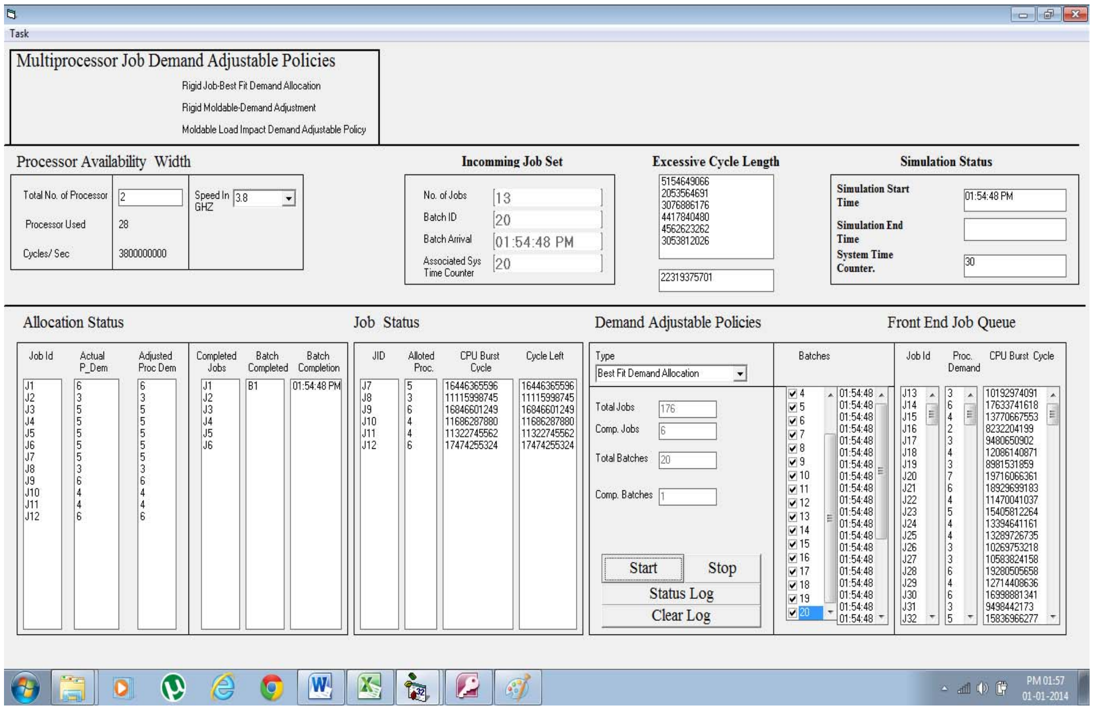

Start
Set JIndex←0
Set JID ← Front_End_job_queue
Cur_Dem ← Get_Pr_Demand(JID)
Monitoring Job Completion and dynamic parallelism, change
Scheduler Continues monitoring Control
Curr_Avail← Get_AvailPr( )
Processor Free List Maintenance thread
Curr_Dem <= Curr_Avail
SetMode(Wait, JID)
Allocate(JID,Curr_Dem)
Set Curr_Avail ← Curr_Avail – Curr_Dem
Set JIndex←JIndex+1

Figure 3.3: Flow chart showing working of strictly Demand Fit Allocation (SDF)



Multiprocessor Job Demand Adjustable Policies

Rigid Job-Best Fit Demand Allocation

Rigid Moldable-Demand Adjustment

Moldable Load Impact Demand Adjustable Policy

Processor Availability Width

Total No. of Processor	2
Processor Used	28
Cycles/ Sec	3800000000
Speed In GHZ	3.8

Incomming Job Set

No. of Jobs	13
Batch ID	20
Batch Arrival	01:54:48 PM
Associated Sys Time Counter	20

Excessive Cycle Length

```
5154649066
2053564691
3076886176
4417840480
4562623262
3053812026
```

22319375701

Simulation Status

Simulation Start Time	01:54:48 PM
Simulation End Time	
System Time Counter.	30

Allocation Status

Job Id	Actual P_Dem	Adjusted Proc Dem	Completed Jobs	Batch Completed	Batch Completion
J1	6	6	J1	B1	01:54:48 PM
J2	3	3	J2		
J3	5	5	J3		
J4	5	5	J4		
J5	5	5	J5		
J6	5	5	J6		
J7	5	5			
J8	3	3			
J9	6	6			
J10	4	4			
J11	4	4			
J12	6	6			

Job Status

JID	Alloted Proc.	CPU Burst Cycle	Cycle Left
J7	5	16446365596	16446365596
J8	3	11115998745	11115998745
J9	6	16846601249	16846601249
J10	4	11686287880	11686287880
J11	4	11322745562	11322745562
J12	6	17474255324	17474255324

Demand Adjustable Policies

Type: Best Fit Demand Allocation

Total Jobs	176
Comp. Jobs	6
Total Batches	20
Comp. Batches	1

Batches	
✓ 4	01:54:48
✓ 5	01:54:48
✓ 6	01:54:48
✓ 7	01:54:48
✓ 8	01:54:48
✓ 9	01:54:48
✓ 10	01:54:48
✓ 11	01:54:48
✓ 12	01:54:48
✓ 13	01:54:48
✓ 14	01:54:48
✓ 15	01:54:48
✓ 16	01:54:48
✓ 17	01:54:48
✓ 18	01:54:48
✓ 19	01:54:48
✓ 20	01:54:48
	01:54:48

Front End Job Queue

Job Id	Proc. Demand	CPU Burst Cycle
J13	3	10192974091
J14	6	17633741618
J15	4	13770667553
J16	2	8232204199
J17	3	9480650902
J18	4	12086140871
J19	3	8981531859
J20	7	19716066361
J21	6	18929699183
J22	4	11470041037
J23	5	15405812264
J24	4	13394641161
J25	4	13289726735
J26	3	10269753218
J27	3	10583824158
J28	6	19280505658
J29	4	12714408636
J30	6	16998881341
J31	3	9498442173
J32	5	15836966277

Figure 3.4: Showing Simulator status strictly Demand Fit Allocation (SDF)

5.2 Extreme-Ending Moldable Approach (EEMA)

Although the final job whose demand does not satisfy, can be moldable to adapt as much as processor space available also referred to as EEMA (Extreme-Ending Moldable Approach). Applicability of such scheduling schemes is only to those systems where overall needed resource request is less than currently available resource limit. Following is the distribution logic

```
For each JID in Front_End_JobQueue
do
    $Set\ Curr\_Dem \leftarrow Get\_Pr\_Demand(JID)$
    $Set\ Curr\_Avail \leftarrow Get\_AvailPr(\,)$
    If  $Curr\_Dem <= Curr\_Avail$  then
        $SetMode(Active, JID)$
        $Allocate(JID, Curr\_Dem)$
        $Set\ Curr\_Avail \leftarrow Curr\_Avail - Curr\_Dem$
        $Set\_AvailPr(Curr\_Avail)$
```

```
else
    $Set\_Mode(Active, JID)$
    $Set\_Pr\_Demand(JID, Curr\_Avail)$
    $Allocate(JID, Curr\_Avail)$
    $Set\ Curr\_Avail \leftarrow 0$
    $Set\_Avail(Curr\_Avail)$
endif
```

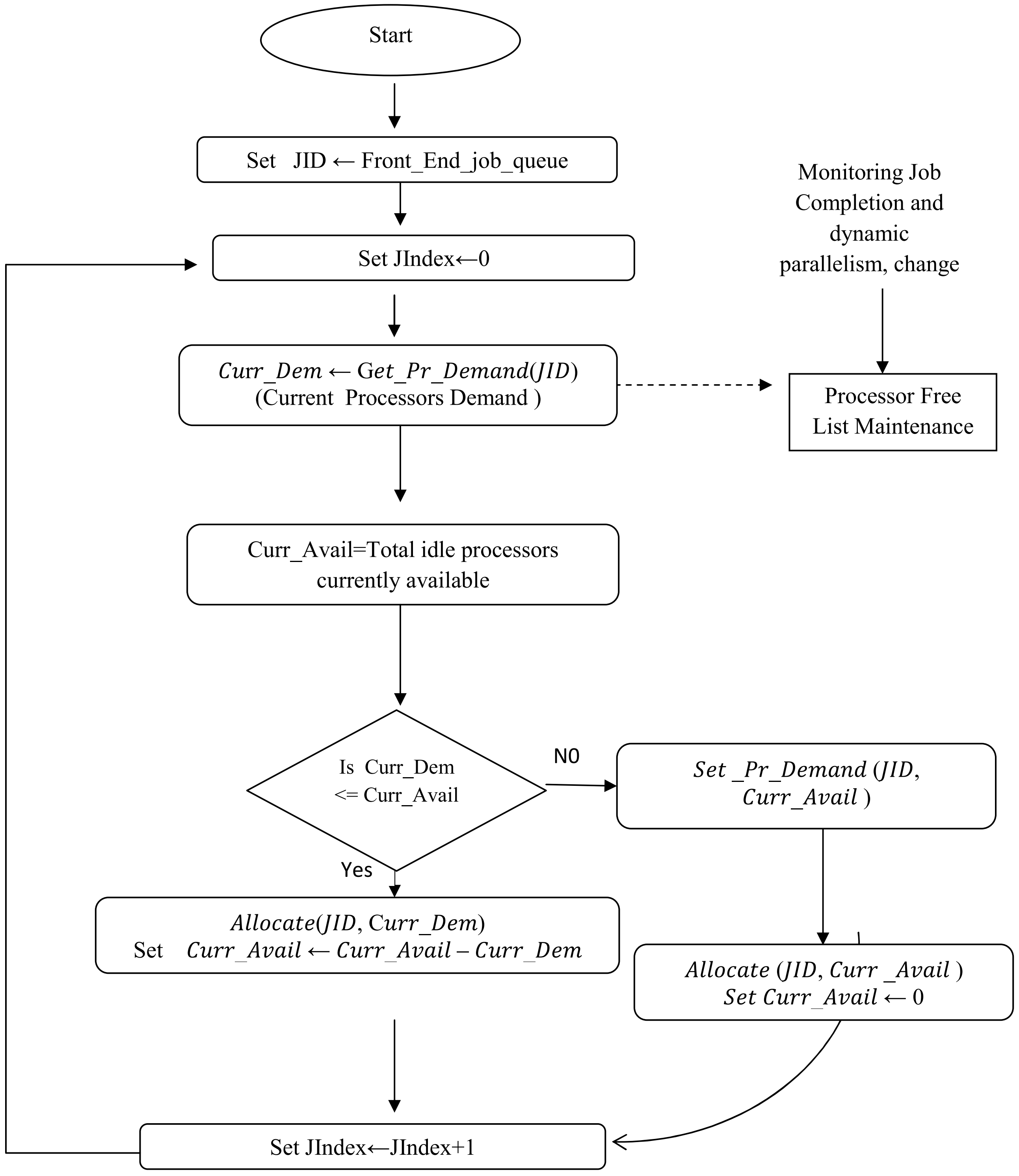

Start
Set JID ← Front_End_job_queue
Set JIndex←0
Monitoring Job Completion and dynamic parallelism, change
Curr_Dem ← Get_Pr_Demand(JID)
(Current Processors Demand)
Processor Free List Maintenance
Curr_Avail=Total idle processors currently available
Is Curr_Dem <= Curr_Avail
N0
Set _Pr_Demand (JID, Curr_Avail)
Yes
Allocate(JID, Curr_Dem)
Set Curr_Avail ← Curr_Avail – Curr_Dem
Allocate (JID, Curr _Avail)
Set Curr_Avail ← 0
Set JIndex←JIndex+1

Figure 3.5: Flow chart showing working of Extreme-Ending Moldable Approach (EEMA)

5.3 Moldable-Load Impact Demand Adjustment (M-LIDA)

Normally a parallel application is designed for a particular processor characteristics, onto which when executed gives tremendous performance but usually architecture employed for execution is not best satisfied to their configuration needs. So such applications are required to adjust their processing demand based on POS (Processor offered space). In other words jobs are converted to moldable while considering what is to be required and what is to be offered. This scheme is referred to as a M-LIDA (Moldable-Load Impact Demand Adjustment) i.e. present load of job and required configuration is adjusted to offered processor space.

For each JID in Front_End_Job Queue

```
do
    $Set\ Curr\_Dem \leftarrow Get\_Pr\_Demand(JID)$
    $Set\ Syn\_WL\_Freq \leftarrow Get\_Pr\_Freq(JID)$
    $Set\ Avail\_Pr\_Freq \leftarrow Get\_Pr\_\text{Avail}\_Freq()$

    $Set\ Adj\_Dem = \frac{Curr\_Dem\ \times Syn\_W\_Pr\_Freq}{Avail\_Pr\_Freq}$

    $Set\_Adj\_Pr\_Demand(JID, Adj\_Dem)$
    If   Adj_Dem <= Curr_Avail   then
        $SetMode(Active, JID)$
        $Allocate(JID, Adj\_Dem)$
        $Set\ Curr\_Avail \leftarrow\ Curr\_Avail - \ Adj\_Dem$
        Set_AvailPr(Curr_Avail)
    else
        $SetMode(Active, JID)$
        $Allocate(JID, Curr\_Avail)$
        $Set\ Curr\_Avail \leftarrow\ 0$
        Set_Avail(Curr_Avail)
    endif
endfor
```

Get_Pr_Demand is software routine for getting current demand of a specified JID. Similarly Get_Avail_Pr computes no. of processor currently available till current allocation barrier. Job can have either active or waiting status mode. Active status is which is ready for dispatching after fulfilling all of its processing requirements. Allocate and Set Avail routines corresponds to the job allocation and setting up available processor space respectively. Demand of any job can be adjusted depending upon conditional construct using Set_Pr_Demand. This routine is only executed during moldable approaches. Synthetic workload as described virtually generated dummy job structures based upon pre-determined processor frequency. Although such type of processors may or may not be available during actual execution. Job demand can be adjusted based upon synthetic and actual available processor frequency, above approaches defined on the basis of space sharing policy mechanism. Allocation is performed on the basis of simultaneous processing thread available for each job, and then the job space will be divided among processor space.

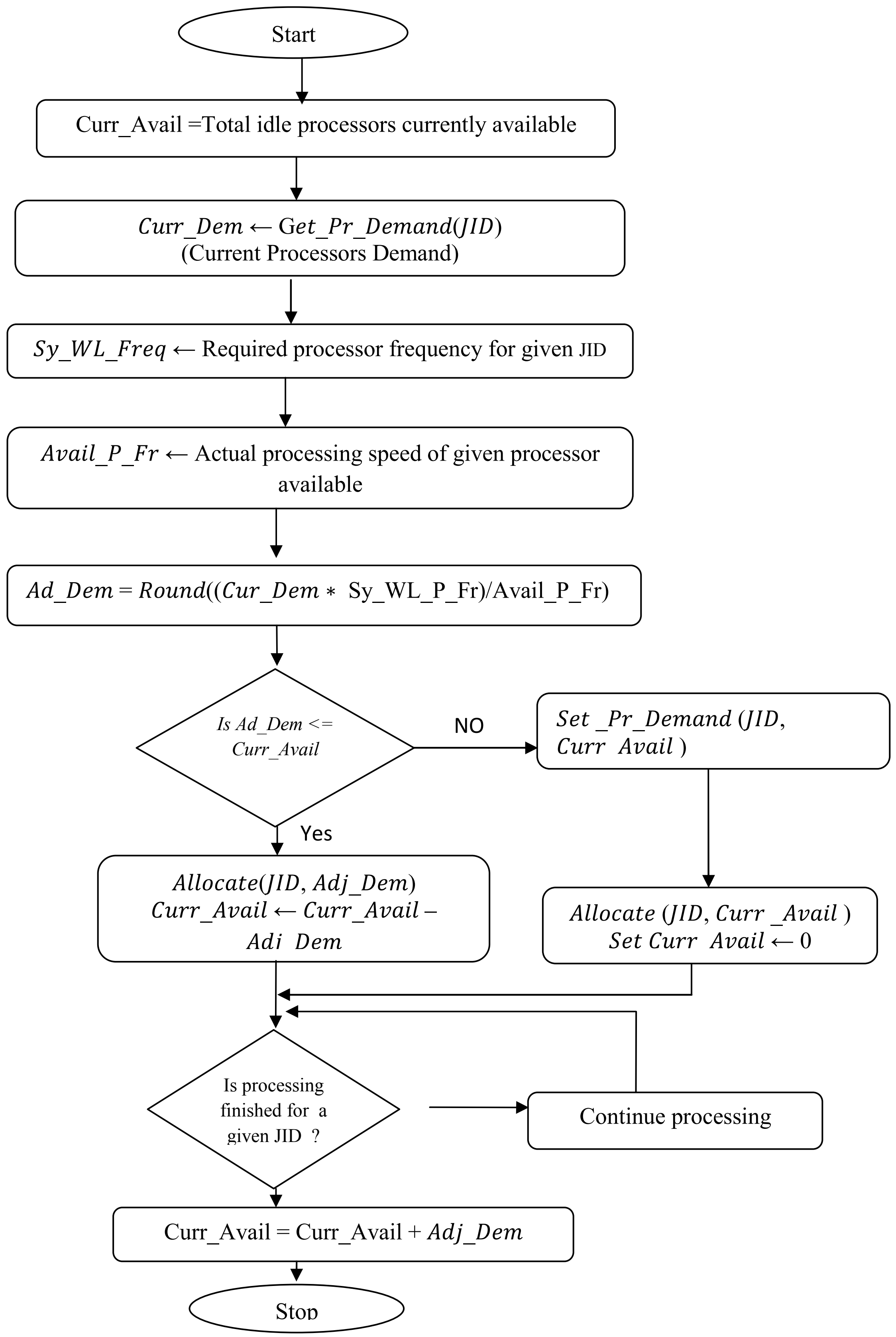
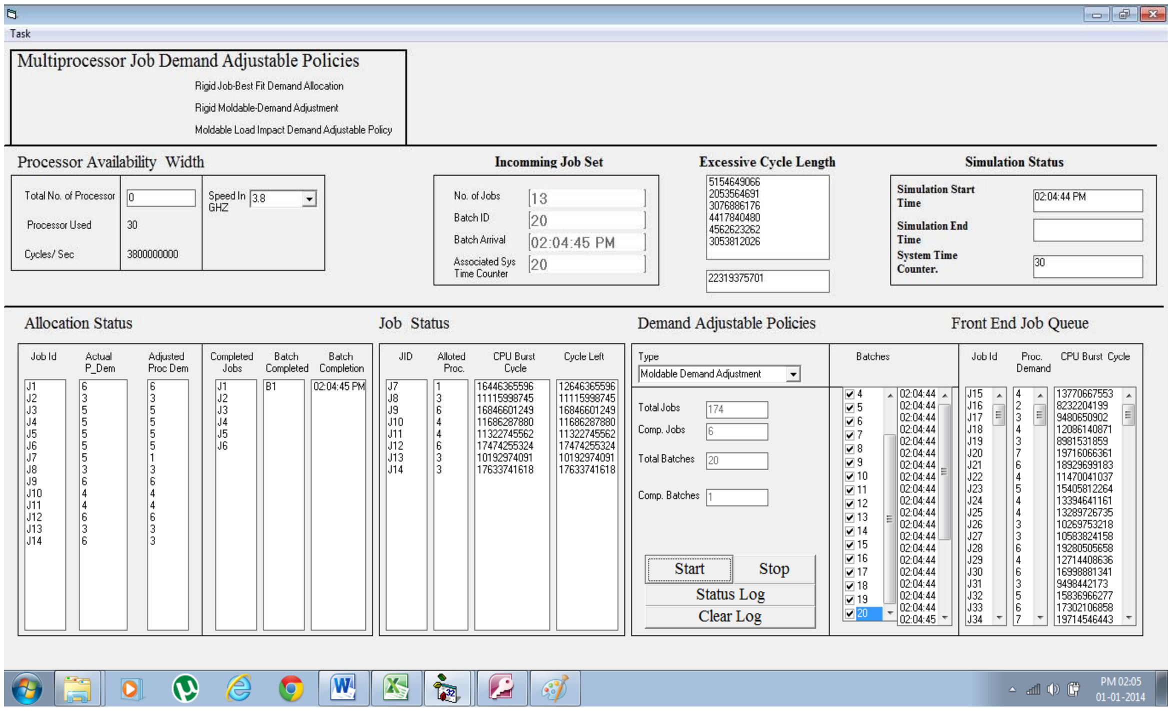

Start
Curr_Avail =Total idle processors currently available
$Curr\_Dem \leftarrow$ Get$_Pr_Demand(JID)$
(Current Processors Demand)
$Sy\_WL\_Freq \leftarrow$ Required processor frequency for given JID
$Avail\_P\_Fr \leftarrow$ Actual processing speed of given processor available
$Ad\_Dem = Round((Cur\_Dem *$ Sy_WL_P_Fr)/Avail_P_Fr)
Is Ad_Dem <= Curr_Avail
NO
$Set\ \_Pr\_Demand\ (JID,$ $Curr\ \ Avail\ )$
Yes
$Allocate(JID, Adj\_Dem)$
$Curr\_Avail \leftarrow Curr\_Avail -$ $Adj\ \ Dem$
$Allocate\ (JID, Curr\ \_Avail\ )$
$Set\ Curr\ \ Avail \leftarrow 0$
Is processing finished for a given JID ?
Continue processing
Curr_Avail = Curr_Avail + Adj_Dem
Stop

Figure 3.6: Flow chart for working of Moldable-Load Impact Demand Adjustment (M-LIDA)



Multiprocessor Job Demand Adjustable Policies

Rigid Job-Best Fit Demand Allocation

Rigid Moldable-Demand Adjustment

Moldable Load Impact Demand Adjustable Policy

Processor Availability Width

Total No. of Processor	0	Speed In GHZ 3.8
Processor Used	30	
Cycles/ Sec	3800000000	

Incomming Job Set

No. of Jobs	13
Batch ID	20
Batch Arrival	02:04:45 PM
Associated Sys Time Counter	20

Excessive Cycle Length

```
5154649066
2053564691
3076886176
4417840480
4562623262
3053812026
```

22319375701

Simulation Status

Simulation Start Time	02:04:44 PM
Simulation End Time	
System Time Counter.	30

Allocation Status

Job Id	Actual P_Dem	Adjusted Proc Dem	Completed Jobs	Batch Completed	Batch Completion
J1	6	6	J1	B1	02:04:45 PM
J2	3	3	J2		
J3	5	5	J3		
J4	5	5	J4		
J5	5	5	J5		
J6	5	5	J6		
J7	5	1			
J8	3	3			
J9	6	6			
J10	4	4			
J11	4	4			
J12	6	6			
J13	3	3			
J14	6	3			

Job Status

JID	Alloted Proc.	CPU Burst Cycle	Cycle Left
J7	1	16446365596	12646365596
J8	3	11115998745	11115998745
J9	6	16846601249	16846601249
J10	4	11686287880	11686287880
J11	4	11322745562	11322745562
J12	6	17474255324	17474255324
J13	3	10192974091	10192974091
J14	3	17633741618	17633741618

Demand Adjustable Policies

Type: Moldable Demand Adjustment

Total Jobs	174
Comp. Jobs	6
Total Batches	20
Comp. Batches	1

Batches	
✓ 4	02:04:44
✓ 5	02:04:44
✓ 6	02:04:44
✓ 7	02:04:44
✓ 8	02:04:44
✓ 9	02:04:44
✓ 10	02:04:44
✓ 11	02:04:44
✓ 12	02:04:44
✓ 13	02:04:44
✓ 14	02:04:44
✓ 15	02:04:44
✓ 16	02:04:44
✓ 17	02:04:44
✓ 18	02:04:44
✓ 19	02:04:44
✓ 20	02:04:44
	02:04:45

Front End Job Queue

Job Id	Proc. Demand	CPU Burst Cycle
J15	4	13770667553
J16	2	8232204199
J17	3	9480650902
J18	4	12086140871
J19	3	8981531859
J20	7	19716066361
J21	6	18929699183
J22	4	11470041037
J23	5	15405812264
J24	4	13394641161
J25	4	13289726735
J26	3	10269753218
J27	3	10583824158
J28	6	19280505658
J29	4	12714408636
J30	6	16998881341
J31	3	9498442173
J32	5	15836966277
J33	6	17302106858
J34	7	19714546443

PM 02:05
01-01-2014

Figure 3.7: Showing Simulator status Moldable-Load Impact Demand Adjustment (M-LIDA)

5.4 Proportionate Processor Width Partitioning (PWP)

Dynamic approach for processor allocation is usually applicable where (NOPA) no. of processors available are less than the no. of processors required (NOPR) by a batch of jobs. Simultaneously, occurred jobs will be grouped under batches. So ultimately the idea is to allocate complete batch regardless of no. of jobs within that batch and their respective demand. So availability should be best adjusted to currently ready batch even the total batch demand is greater than the currently offered processor space. If processor offered space POS is more than that of PRQ processor required space than any of the above defined approaches can be employed but proportionate scheme is applicable to batch oriented systems where POS is least than required, the best adjustment will be performed.

$$Total\_Dem\_Batch_j = \sum_{i=1}^{n} Job\_Dem_i$$

Where jth is the batch. Proportionate scheduling will be applied only when following two conditions are meet:-

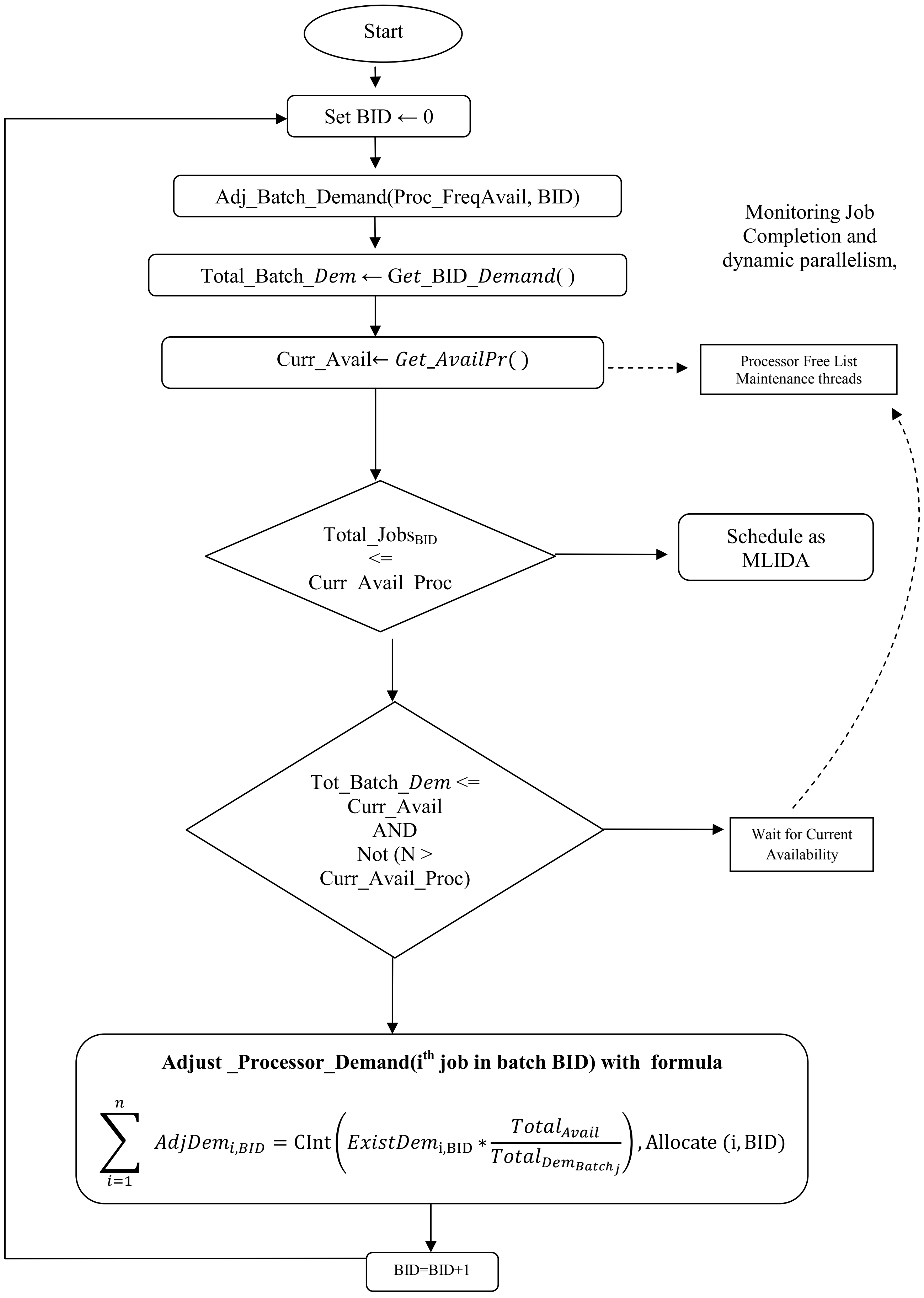
a.) Current availability of processor should be less than or equal to total demand required by the current batch.

b.) Also no. of available jobs in the batch must be less than or equal to current availability of processors.

Otherwise if the current availability is more you can employ any of the above defined policy than proportionate. Basic idea is to active complete batch with best processor managed space.

$$AdjDem_{i,j} = \sum_{i=1}^{n} \mathrm{CInt}\left(ExistDem_{\mathrm{i,j}} * \frac{Total\_Avail}{Total\_Dem\_Batch_j} \right)$$

In this case demand of i^{th} job in j^{th} batch will be adjusted. This process is performed for each job in the batch at once. During each adjustment total availability as well as total batch demand will be changed.


Start
Set BID ← 0
Adj_Batch_Demand(Proc_FreqAvail, BID)
Total_Batch_Dem ← Get_BID_Demand( )
Curr_Avail← Get_AvailPr( )
Monitoring Job Completion and dynamic parallelism,
Processor Free List Maintenance threads
Total_Jobs_BID <= Curr Avail Proc
Schedule as MLIDA
Tot_Batch_Dem <= Curr_Avail AND Not (N > Curr_Avail_Proc)
Wait for Current Availability
Adjust _Processor_Demand(ith job in batch BID) with formula
BID=BID+1

$$\sum_{i=1}^{n} AdjDem_{i,BID} = \mathrm{CInt}\left(ExistDem_{\mathrm{i,BID}} * \frac{Total_{Avail}}{Total_{Dem_{Batch_j}}}\right), \mathrm{Allocate\ (i, BID)}$$

Figure 3.8: Flow chart for working of Proportionate Processor Width Partitioning (PWP)

3.4 Demand Adjustment Benefits

Benefits for demand adjustment in (M-LIDA) before allocation provides tremendous run-time change in ongoing parallelism. Other policy structures, where major motivation is on allocation without considering resource adjustment mapping against given load are follows static aspects of workload assignment. Executes jobs only when corresponding required resource configuration is available. Advantage of resource/demand adjustment (lower to higher frequency) against given load leads to the increased no. of currently active scheduling jobs, increases PLP (Process Level Parallelism). Another advantage of demand adjustment policies irrespective to frequency adjustment, If the processor offered space (POS) is available more in comparison to previously executed scenario ultimately the currently active jobs will also increases. So in demand adjustment either the POS value is increased or higher frequency processors are available than synthetic one, the result will be increased no. of active job as described in fig-3.10. However, in demand adjustment if lower frequency processors are available than required the result will be increase in job's demand and ultimately the currently active list will be dependent upon POS value available as shown in the fig–3.10. For example if synthetic workload processor frequency is of 2.0 Ghz and the system has only 20 (POS) processors available of 1Ghz each. Now if a job occurred having demand 5 ultimately during adjustment his demand will be adjusted to 10 leading to occupying half of the POS space. So automatically affected to currently active jobs sets i.e. decreased Process level parallelism (PLP). Despite of this, where demand adjustment against load with respect to running processor frequency is not considered, currently active jobs will be increased only when processor offered space is increased rather than their frequency clock as described in Figure 3.9. Also these policies leads to higher Excessive cycle length (processor cycle wastage) as compare to demand adjustment policies described later.

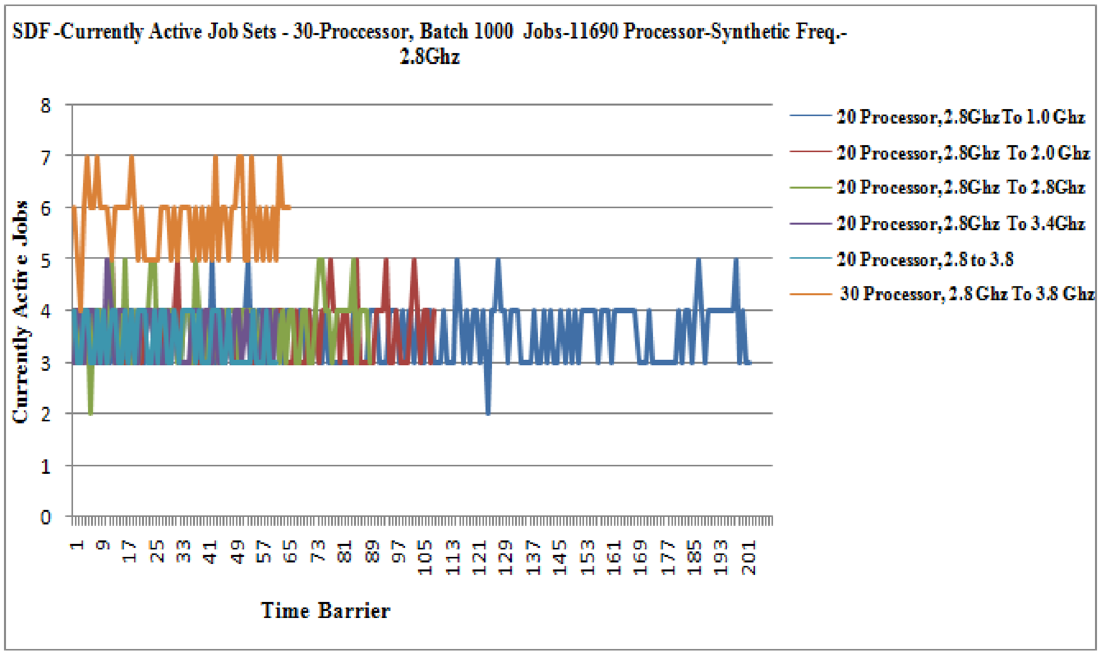


Figure 3.9: Currently Active Jobs in SDF

As in the above graph, Strictly Demand Fit policy has been applied, as the no. of processor increased without increase in the processor frequency, the currently active jobs are more. This provides the benefits over increase in frequency, so no matter what the speed of the processor is – only the key issue is how much POS (processor offered space) is. But in load adjustment policy (M-LIDA) where either frequency is increased or POS value is increased, the currently jobs set is increased automatically. Because after load adjustment, POS value will be more than PRS (processor required space).

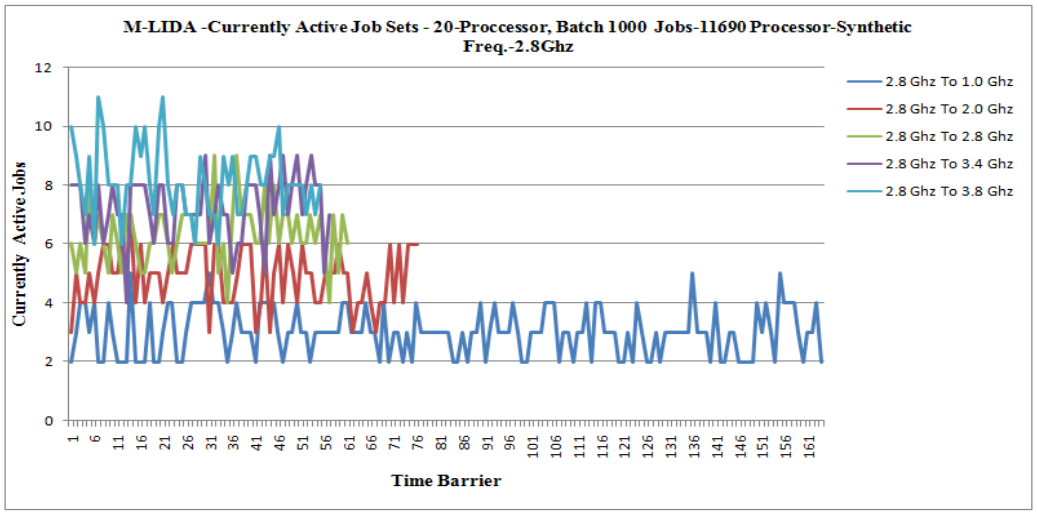


Figure 3.10: Currently Active Jobs in M-LIDA with POS-20

3.5 Run Time Demand Adjustments

During each periodic time barrier, Load adjustment scheduler Performs monitoring each job's remaining scheduling cycles with respect to no. of processor allocated and their respective frequency. If possible the demand will be adjusted; there are two cases of this dynamic demand change management. If the processors are allocated approximately near to the required demand like in M-LIDA, the demand will decreases during further completion stages, there is no case available in which demand will increases. Another case, which is occurred in proportionate allocation where demand is already set to minimum calculated threshold, so in this case the demand might increase or decrease during further adjustment. Following is the formal method for dynamic demand change management.

$$AdjDem_i = \sum_{i=1}^{Act\_Que\_Length} \text{Round}\left(\frac{RemRemCycle_{\text{i}}}{Allo\_\text{Pr}\ \_Frq} \right)$$

Chapter-4
Results and Discussion

4.1 Parameters under consideration

Simulation produced while considering above defined policies takes synthetic workload as input. Several parameters have been studied and considered for constructing illustrations. Factors and their respective detail is as follows-

i. **Currently Active Jobs**: This parameter defines currently active jobs by managing current processor availability. If more no. of currently active job running, then process level parallelism will be increased i.e frequent response from the system to many no. of parallel users.

ii. **Excessive Cycle Length (ECL):** This parametric value defines processor allocated in excess than required. As the job reduces its burst during its execution life cycle its processor requirement will be reduced. As the job execution life cycle reaches at its final stage, the length of ECL will increase. Although, processor demand can be changed to current required but at final stage adjusting demand will be more costly than continuous execution with the previous allocation. Changing job demand will pause its execution. Time consumed for adjustment might be more expensive than consumed with previous defined allocation. There may be the possibility that job's final execution level will be completed within that time. For example:

- Suppose a particular job requires 5 processor of frequency 2.8 Ghz with CPU burst 148674578647 .But available processors are of 3.8 Ghz

  Total number of cycles available= $3.8*10^9$ *5=19000000000

  ECL=19000000000-14867457867=4132542133

This is the amount of excessive cycle wastage which has occurred as the available processor is of higher frequency (3.8) than the required frequency (2.8) and we did not calculate the actual requirement based upon different configuration available. In the thesis based upon the different policies discussed, the ECL is calculated. ECL for SDF came to be maximum in case of EEMA and minimum for PWP.

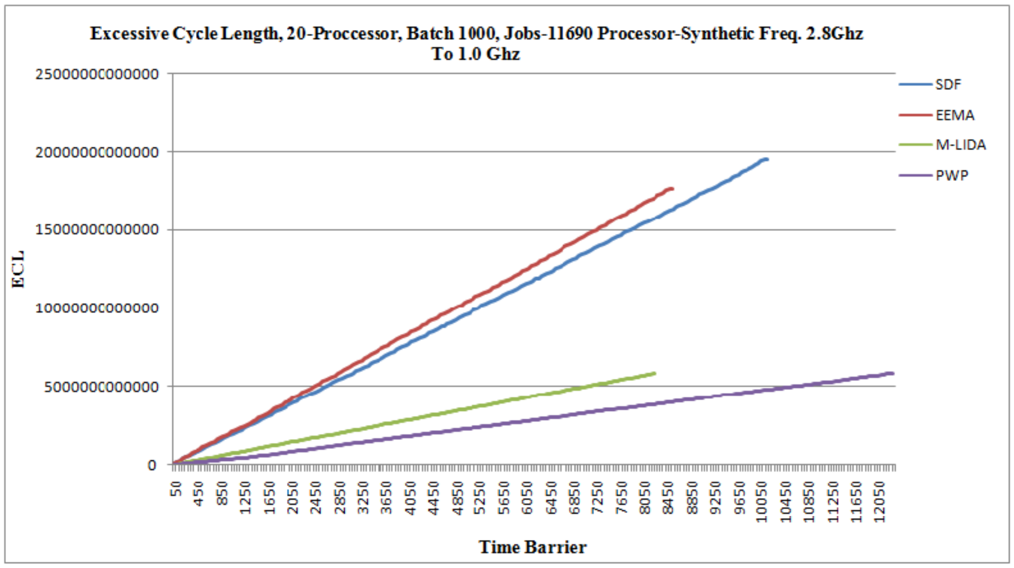


Figure 4.1: Excessive cycle Length

iii. **Overall Execution End Time (OEET)**: This parameter defines the maximum simulation time (sec) to compute given no. of batches. Although this parameters is a static one because if the requirement is to manage processor offered space along with increased currently active job set, then this factor may increase overall simulation time.

iv. **Total N–Scheduled Job**: Total no. of scheduled jobs up to a given time barrier is another factor to evaluate simulation efficiency in terms of throughput. This will combine no. of completed jobs along with no. of currently active jobs.

v. **Processor Utilization Per Process**: This term can be defined as *processor managed space at process level/thread level.* If the thread level parallelism is increased more no. of processors are allocated to single active job to handle MSPT (multiple simultaneous processing threads). Ultimately the current active job set will be decreased. Performing best processor space management jobs are converted to moldable/malleable structure [9] [10], this will increase process level parallelism.

vi. **No. of Completed Jobs_**– This term can be defined as actual throughput obtained at any given timer barrier. Only no. of completed jobs are evaluated at each timer barrier as-

$Total\_Comp\_Jobs = Total\_N\_Sched - Curr\_Act\_Jobs$

4.2 Experimental Setup

Simulation based proposed implementation is developed using Visual Basic Studio 6.0 in order to evaluate the proposed moldable scheduling algorithms under various constraints/parameters. Most of the cases the simulation modelling can provide us more generalized results which are most promising as compared to actual implementation on hardware. Further the evaluation of algorithms over a broad range of characterization like changing number of processor requirements of each job, varying processor frequency for each job, varying CPU burst time for each job etc. Job queues associated with the logically programmed virtual processors are used in order to implement various scheduling policies. The simulation environment consists of synchronized communication using various thread timers controls available in language. The setup consists of number of batches arrived using random distribution covering workload aspects such as unique Batch_ID, each Batch_Id consists of its associated job list having unique Job_Id, CPU cycle burst required, processor demand etc. CPU burst cycle specifies the length of execution of a particular job. As the cycle length increases the amount of time of execution will also increases. The cycle capacity of each processor will vary according to the operational clock frequency of that particular processor. Characteristics of the simulation environment:-

i. **Graphical user interface**: It helps in ease to use and understanding of how the things are working in a user friendly manner.

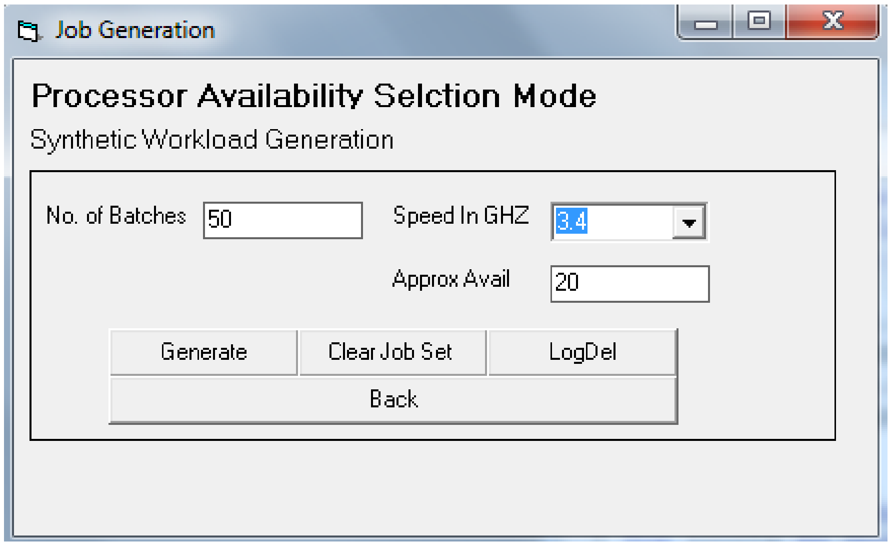
ii. **Different menus**: The simulator has different buttons for different purposes like generation of the work load with required number of processors, various frequency selection options through drop down list. There is also an option through which we can clear the entire database with just a single click. So that any inconsistency during data collection can be avoided. For taking snapshot of the database at a particular time there is also Start/Stop button.

iii. **Data storage**: Data generated by applying various policies is stored in MS-Access.

iv. **Status Window**: It tells the current status of the simulator.

v. **Various checkboxes drop down lists, text boxes**: All these toolkits helps in giving inputs, data capturing generated during the execution.

vi. **Performance measurement during execution**:-The text boxes for simulation start time, end time, excessive cycle length etc. helps in measuring run time status at various intervals.

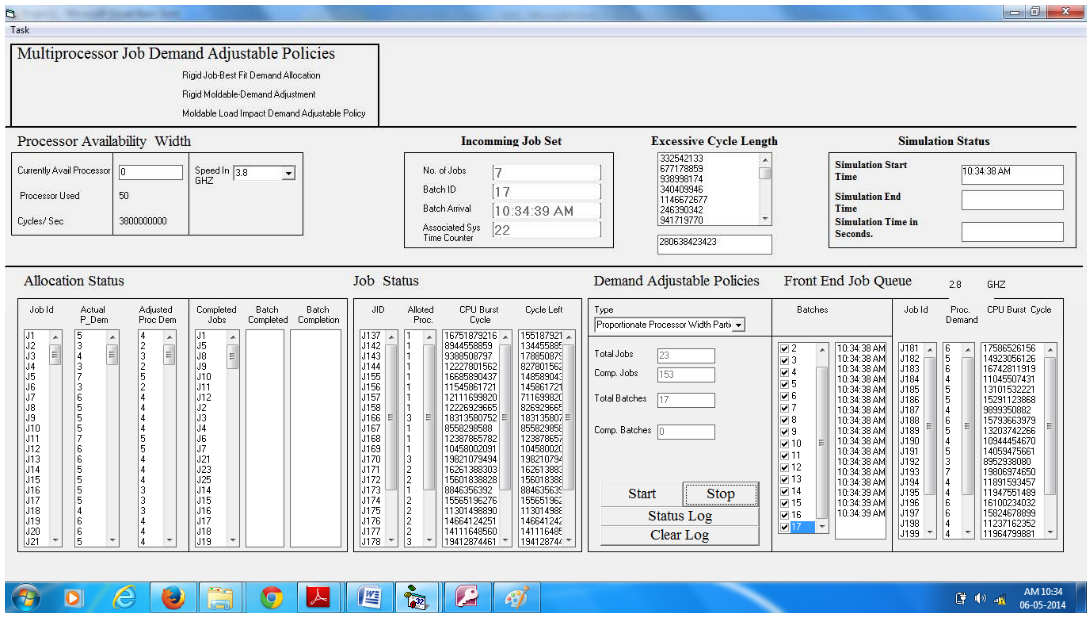


Processor Availability Selction Mode

Synthetic Workload Generation

No. of Batches	50
Speed In GHZ	3.4
Approx Avail	20

Figure 4.2: Showing layout of load generation



Multiprocessor Job Demand Adjustable Policies

Rigid Job-Best Fit Demand Allocation

Rigid Moldable-Demand Adjustment

Moldable Load Impact Demand Adjustable Policy

Processor Availability Width

Currently Avail Processor	0	Speed In GHZ	3.8
Processor Used	50		
Cycles/ Sec	3800000000		

Incomming Job Set

No. of Jobs	7
Batch ID	17
Batch Arrival	10:34:39 AM
Associated Sys Time Counter	22

Excessive Cycle Length

```
332542133
677178859
938998174
340409946
1146672677
246390342
941719770
```

280638423423

Simulation Status

Simulation Start Time	10:34:38 AM
Simulation End Time	
Simulation Time in Seconds.	

Allocation Status

Job Id	Actual P_Dem	Adjusted Proc Dem	Completed Jobs	Batch Completed	Batch Completion
J1	5	4	J1		
J2	3	2	J5		
J3	4	3	J8		
J4	3	2	J9		
J5	7	5	J10		
J6	3	2	J11		
J7	6	4	J12		
J8	5	4	J2		
J9	5	4	J3		
J10	5	4	J4		
J11	7	5	J6		
J12	6	5	J7		
J13	6	4	J21		
J14	5	4	J23		
J15	5	4	J25		
J16	5	3	J14		
J17	5	3	J15		
J18	4	3	J16		
J19	6	4	J17		
J20	6	4	J18		
J21	5	4	J19		

Job Status

JID	Alloted Proc.	CPU Burst Cycle	Cycle Left
J137	1	16751879216	155187921
J142	1	8944558859	134455885
J143	1	9388508797	17885087
J144	1	12227801562	827801562
J155	1	16685890437	14858904
J156	1	11545861721	145861721
J157	1	12111699820	711699820
J158	1	12226929665	826929665
J166	3	18313580752	18313580
J167	1	8558298588	85582985
J168	1	12387865782	12387865
J169	1	10458002091	10458002
J170	3	19821079494	19821079
J171	2	16261388303	16261388
J172	2	15601838828	15601838
J173	1	8846356392	88463563
J174	2	15565196276	15565196
J175	2	11301498890	11301498
J176	2	14664124251	14664124
J177	2	14111648560	14111648
J178	3	19412874461	19412874

Demand Adjustable Policies

Type: Proportionate Processor Width Parti

Total Jobs	23
Comp. Jobs	153
Total Batches	17
Comp. Batches	0

Front End Job Queue 2.8 GHZ

Batches	
✓ 2	10:34:38 AM
✓ 3	10:34:38 AM
✓ 4	10:34:38 AM
✓ 5	10:34:38 AM
✓ 6	10:34:38 AM
✓ 7	10:34:38 AM
✓ 8	10:34:38 AM
✓ 9	10:34:38 AM
✓ 10	10:34:38 AM
✓ 11	10:34:38 AM
✓ 12	10:34:38 AM
✓ 13	10:34:38 AM
✓ 14	10:34:38 AM
✓ 15	10:34:39 AM
✓ 16	10:34:39 AM
✓ 17	10:34:39 AM

Job Id	Proc. Demand	CPU Burst Cycle
J181	6	17586526156
J182	5	14923056126
J183	6	16742811919
J184	4	11045507431
J185	5	13101532221
J186	5	15291123868
J187	4	9899350882
J188	6	15793663979
J189	5	13203742266
J190	4	10944454670
J191	5	14059475661
J192	3	8952938080
J193	7	19806974650
J194	4	11891593457
J195	4	11947551489
J196	6	16100234032
J197	6	15824678899
J198	4	11237162352
J199	4	11964799881

Figure 4.3: Showing different components of simulator

4.3 Illustrations

In case of M-LIDA frequency of processor is increased or decreased rather than their POS value (20), results are better than in SDF policy allocation. In case of SDF results are better only when POS value i.e. 20 to 30 is adjusted. No efficient effect seems to display when frequency is increased in SDF. In spite of this, the results are also better when POS space value is increased in M-LIDA rather than frequency as described in figure below

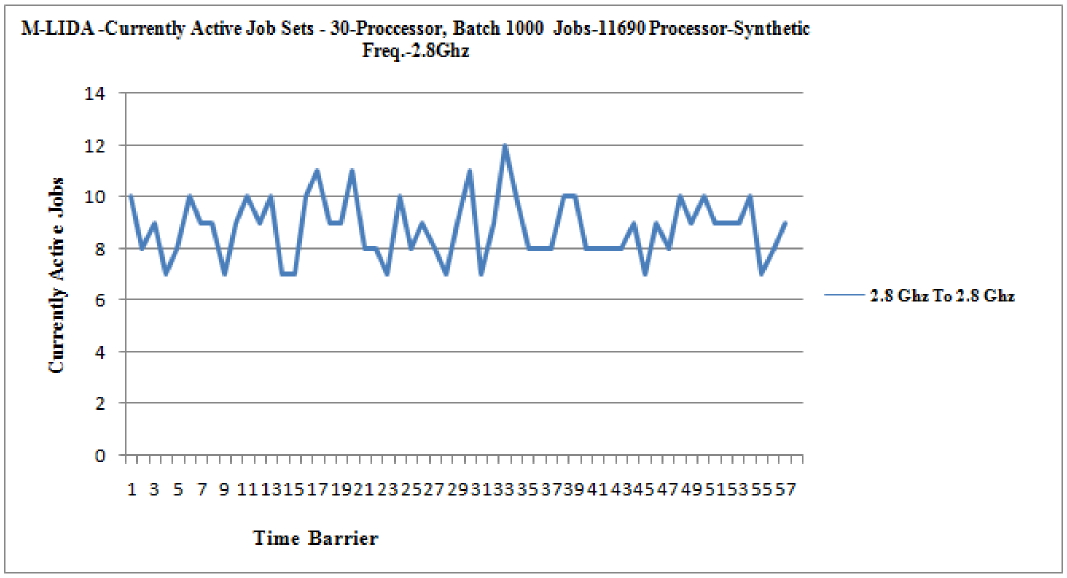


Figure 4.4: Currently Active Jobs in M-LIDA with POS-30.

In proportionate processor width partitioning scheme, POS space will be managed as minimum as possible so always increase in currently active list, although delay may be in final job completion. So ultimately whole simulation will end by consuming much of the time than other policy execution as describe in the below illustrations.

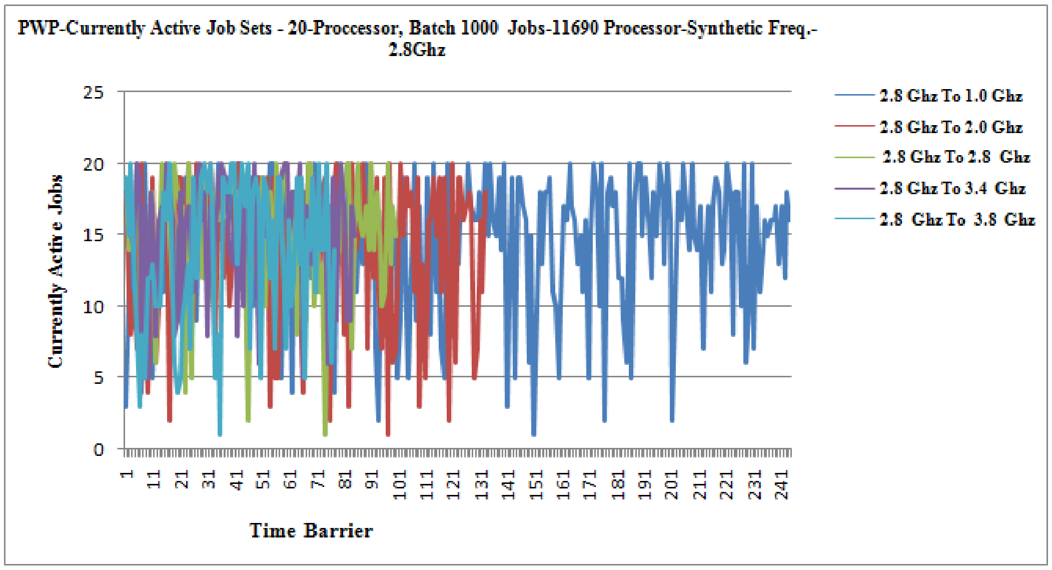


Figure 4.5: PLP (Process Level Parallelism) - in PWP.

Consider the above PWP proportionate processor width partitioning graph for currently active job. The policy will have long execution cycle. Although currently active job set will be increased leading to more process-level parallelism as compare to thread level parallelism. This also leads to more delay within overall execution completion i.e. jobs completion time will be more. This is because the processors are allocated less as compare to requested demand. The current state of affairs shows that policy is applicable where focus is on increased multiple user response required i.e. more no. of users are responded at given time barrier. Process level parallelism also takes care of job's initiation time i.e. jobs are invoked earlier even with less processor scheduled as required. Further illustrations describe demand adjustment gives benefits to where exact mapping of required resources is not performed. The Figure 4.7 to Figure 4.10 exhibits as the frequency is adjusted from lower to higher the result will be better in M- LIDA. Despite of this, if frequency is adjusted from higher to lower then the resource demand will be increased per process, PLP will be decreased as described and thread level parallelism will be increased. Total No. of Completed Jobs considered as a key factor of measuring overall performance in terms of throughput at particular barrier time. Further, the analysis produces throughput effect attained by each of the scheduling structure.

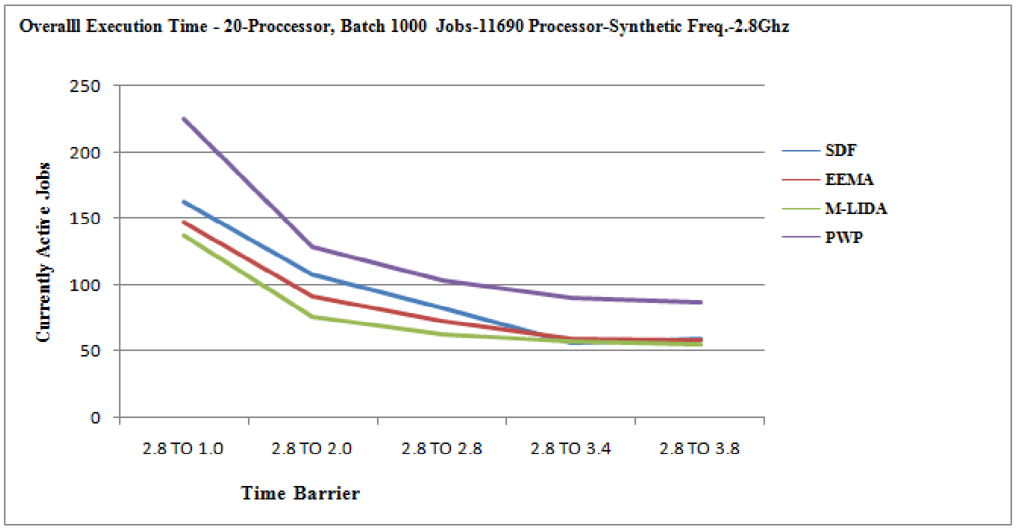


Figure 4.6: Overall Execution End Time

Below are the variation graphs captured at different processor frequency available. Ultimate idea behind this is to illustrate process level parallelism i.e. more currently active jobs. The graphs produced described different policy sets corresponding to discrete processor frequency. Although increased currently active set does not lead to more no. of completed jobs.

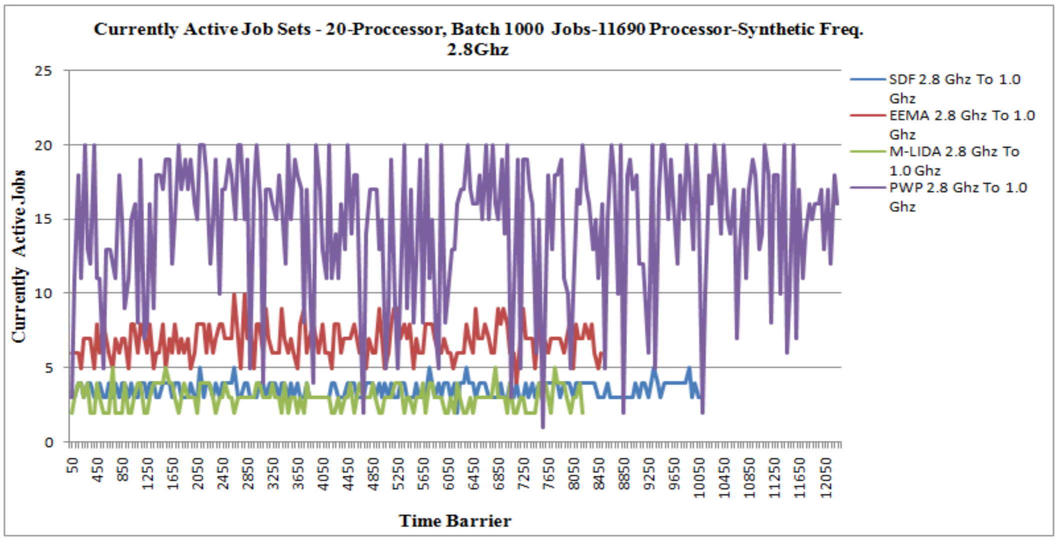


Figure 4.7: PLP (Process Level Parallelism)-2.8 GHz to 1.0 GHz.

This is because as more as the processor managed space, delay in final job completion time end as described above in figure 4.6.

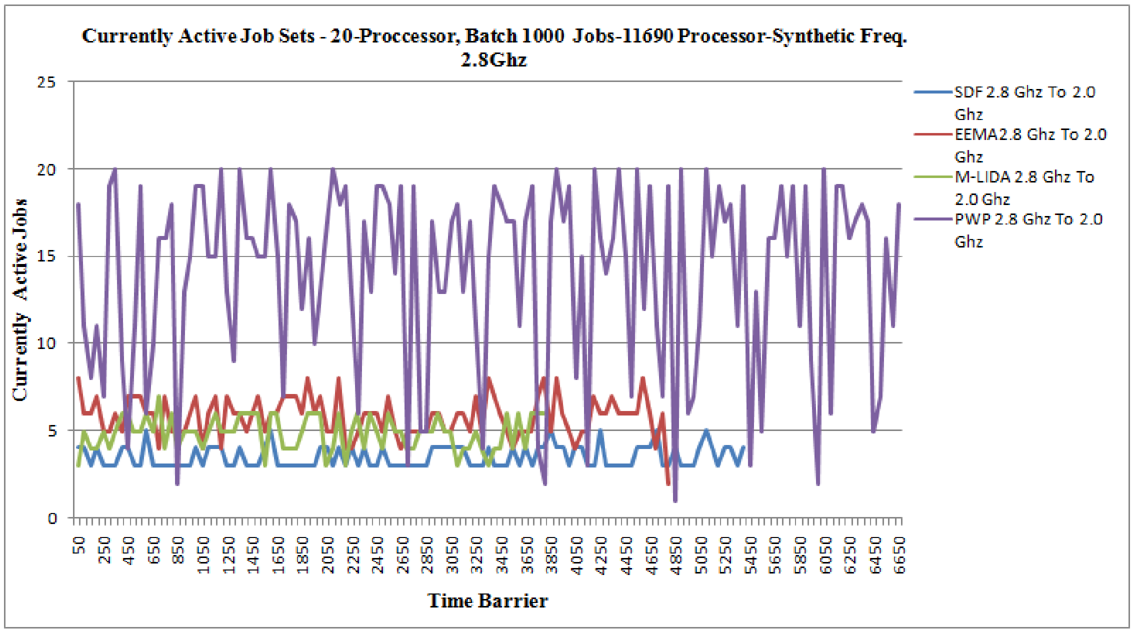


Figure 4.8: PLP (Process Level Parallelism)-2.8 Ghz to 2.0 Ghz.

Excessive cycle length is computed in terms of wastage i.e. extra processor allocated. Processor once allocated if not adjusted due to parallelism change will lead to ECL. M-LIDA Policy structure always monitors ongoing parallelism change and allocated the processor as required by corresponding load remaining. This is required to schedule next batch as earlier as possible without incorporating delay as much as possible.

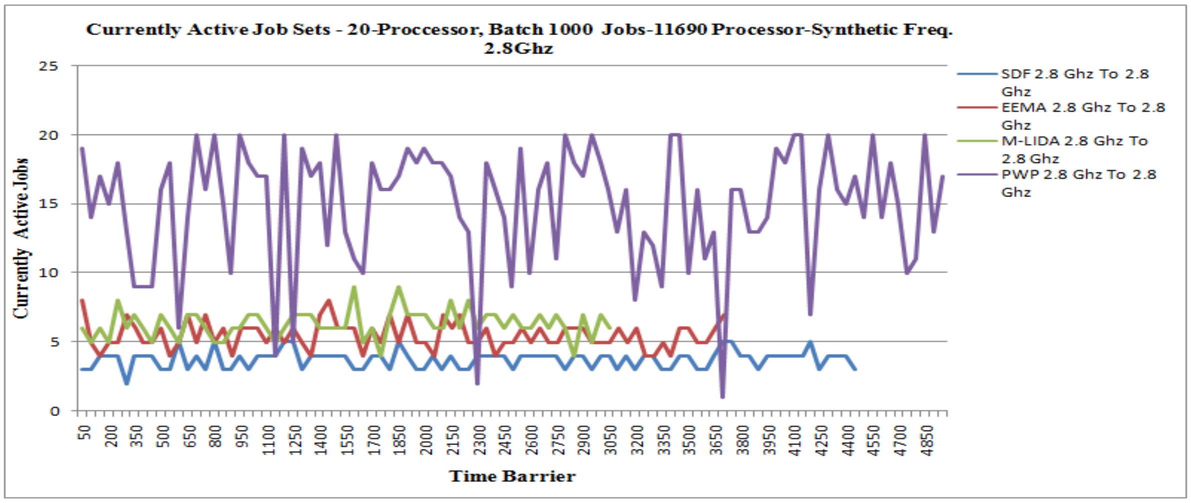


Figure 4.9: PLP (Process Level Parallelism)-2.8 GHz to 2.8 GHz.

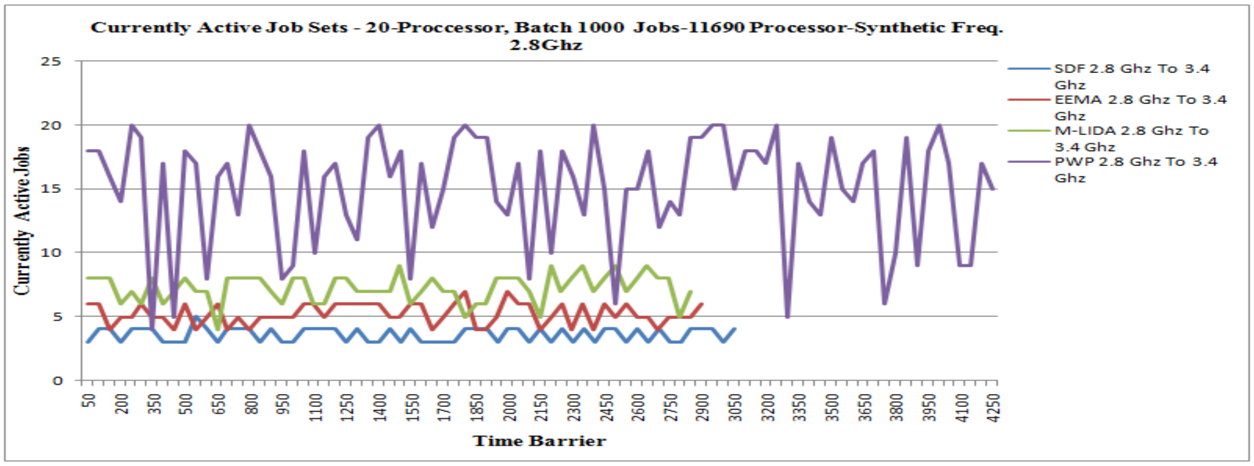


Figure 4.10: PLP (Process Level Parallelism)- 2.8 GHz to 3.4 GHz.

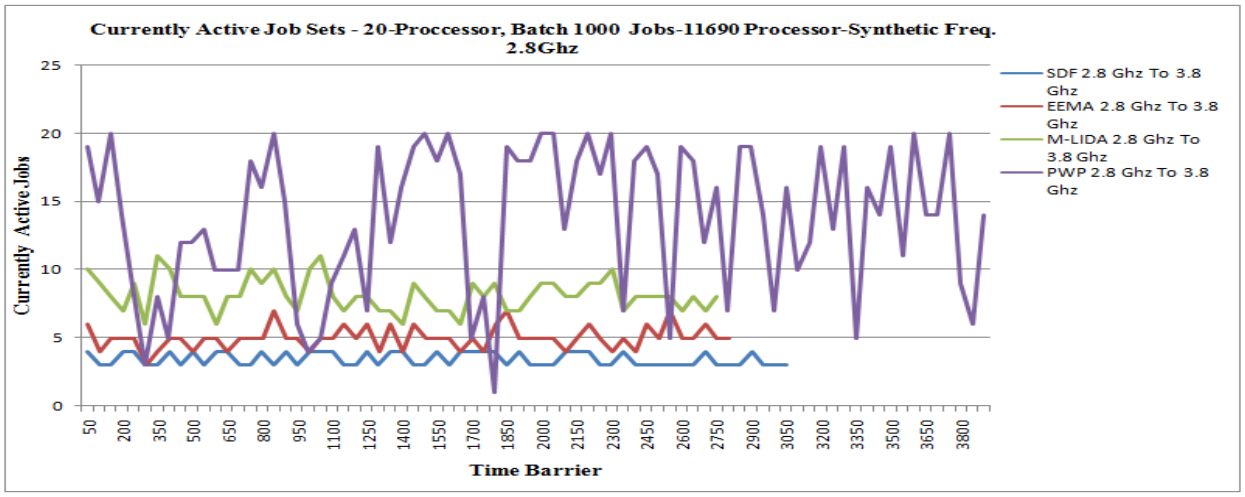


Figure 4.11: PLP (Process Level Parallelism)- 2.8 GHz to 3.8 GHz.

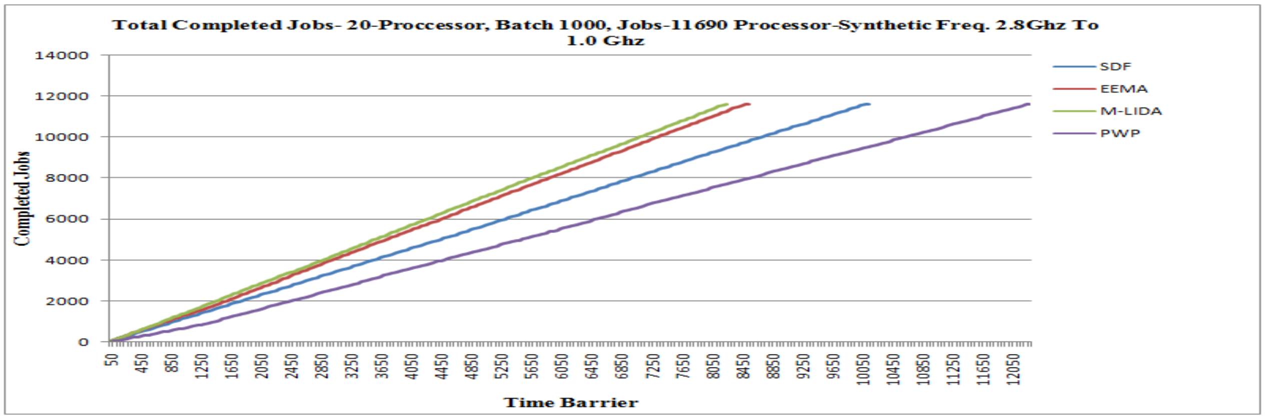


Figure 4.12: Throughput– 2.8 Ghz to 1.0 Ghz.

As described if frequency is adjusted from higher to lower, overall end timer barrier will be more in each of the policy.

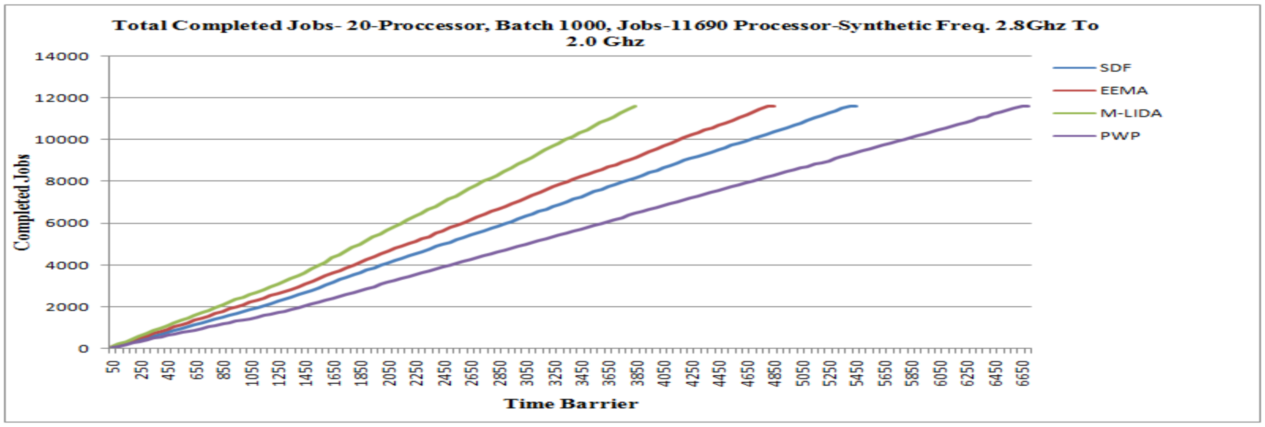


Figure 4.13: Throughput– 2.8 GHz to 2.0 GHz.

No. of completed jobs calculated cumulatively at each barrier is also less as compare to scenarios captured from lower to higher frequency. Policy structures SDF and EEMA where PLP status is less as compare to PWP policy mechanism i.e. TLP thread level parallelism is more in SDF and EEMA, also throughput is more than PWP. These are demand promising approaches i.e. more prone to demand satisfaction.

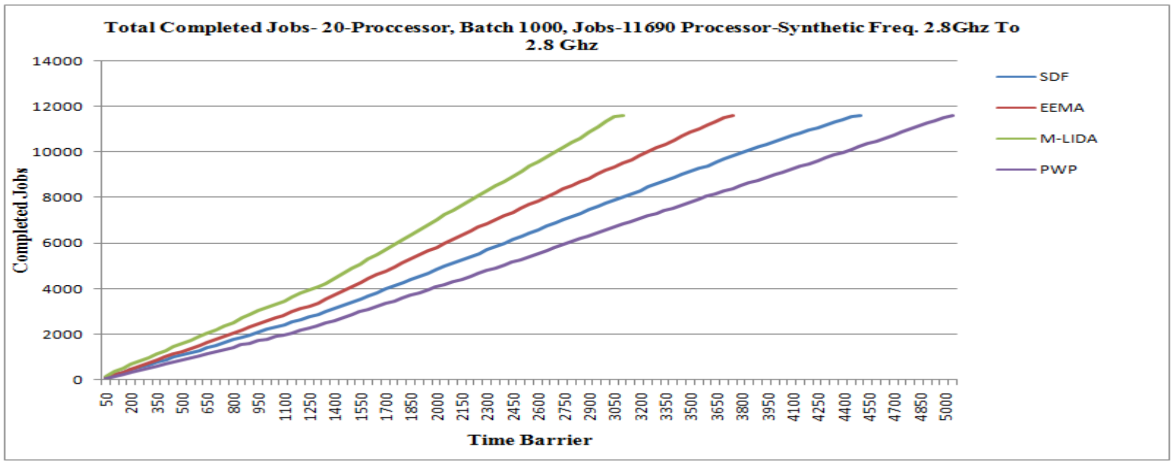


Figure 4.14: Throughput– 2.8 GHz to 2.8 GHz.

In M-LIDA the demand will be adjusted to current need of the job, so remaining allocated processors are placed into free list and are scheduled to next incoming jobs. So ultimately overall end time will be shorter in any case as compare to other, also throughput will be more in all the cases illustrated.

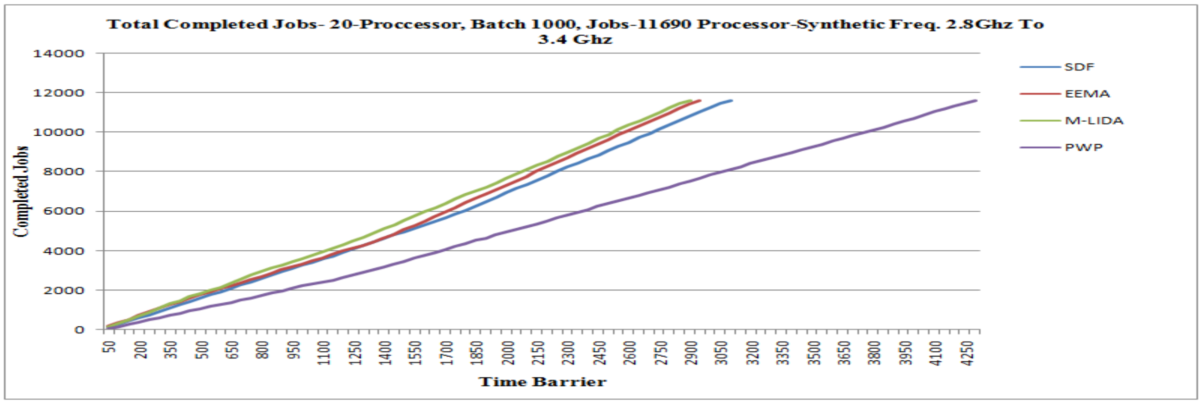


Figure 4.15: Throughput– 2.8 Ghz to 3.4 Ghz.

Consider PWP division where processors are allocated earlier as minimum as possible, later on demand of some jobs might be increased depending upon the processor availability, if decreased will be beneficial and POS space will be more in that case. Two conditions must be met to schedule a complete batch in PWP. Otherwise the remaining POS will be adjusted to currently active jobs to fulfill their processing deeds. This will make delay in next batch allocation

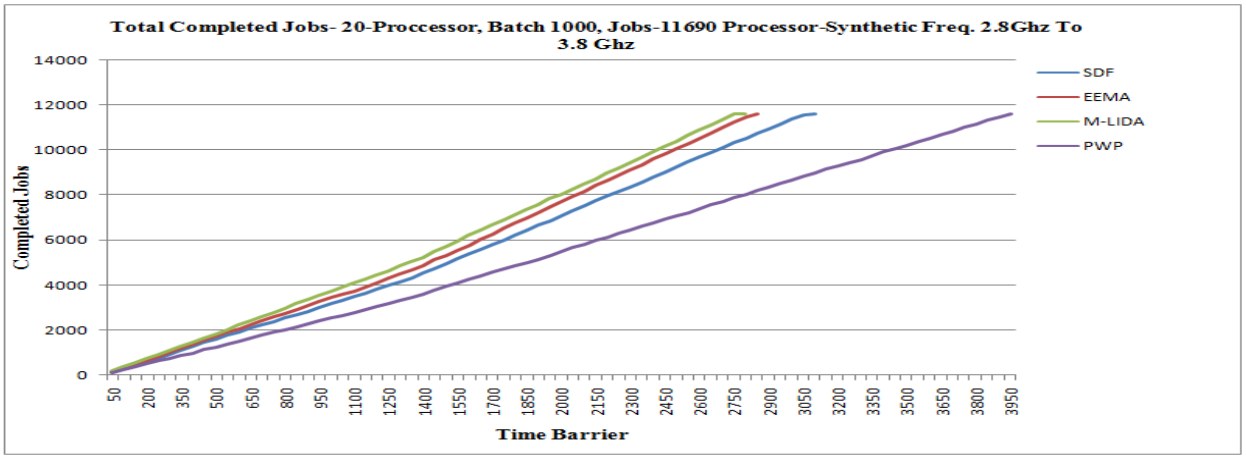


Figure 4.16: Throughput– 2.8 GHz to 3.8 GHz.

The produced illustrations exhibits that for better processor managed space (PMS), demand adjustment is required. Consider PWP where small no. of total jobs are completed against each time barrier as compare to other policy mechanisms, although

PWS has more currently active set. So more number of active jobs produces delay in overall completion as well as delay in average completion time of each job.

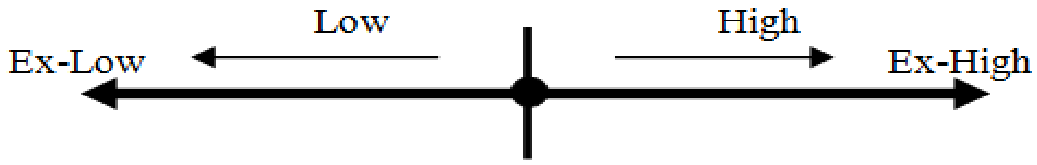
TLP (Thread Level parallelism) in M-LIDA is also very close as compare to demand promising jobs, this is because demand is adjusted depending upon the required cycle burst, demand may be increased or decreased corresponding to the current configuration available. Now ECL excessive cycle length will be demonstrated in further produced illustrations.ECL list is more in demand promising jobs, because focus is on satisfying best possible allocation to each job wherever possible, because of no consideration of processors dynamic characteristics i.e. speed corresponding to job demand

Chapter 5
Conclusions

5.1 Conclusion

Described Illustrations exhibits that M-LIDA and PWP policy mechanism will be best in processor managed space (POS). In addition ECL list is minimum along with incorporating demand promising model (DPM) in M-LIDA. However for increased PLP and minimum priority for overall completion time PWP will be best. The idea behind this implementation research is because in realty there will be a mismatch in required space and available space. So execution over discrete frequency sample provides best selection for real time infinite job execution. Extremely end moldable job will be best that SDF, small amount of change in the logic. At the end, when processor availability does not satisfy the job demand, job will be converted to moldable by allocating as much as processor space available. This will lead to more execution at each time barrier i.e. (n+1) jobs are allocated as compare to SDF. ECL wastage in M-LIDA is due to the final end of the job execution where only single processor allocated and remaining left cycle of job is less than the frequency of that processor. In PWP processors are allocated as minimum as possible so job burst cycles are mostly greater than the frequency of allocated no. of processor of the job. So produces delay in reaching final job's completion end. In addition ECL produces as less as compare to other policy mechanism with a longer length. Following is the behavioural analysis described along with various policies parameters:

Table 4.1:-Showing Summarize behaviour of various parameters for various Schemes



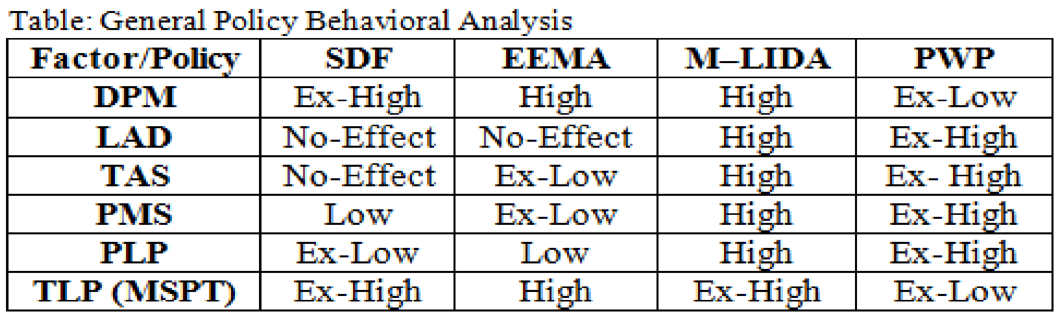


Table: General Policy Behavioral Analysis

Factor/Policy	SDF	EEMA	M–LIDA	PWP
DPM	Ex-High	High	High	Ex-Low
LAD	No-Effect	No-Effect	High	Ex-High
TAS	No-Effect	Ex-Low	High	Ex- High
PMS	Low	Ex-Low	High	Ex-High
PLP	Ex-Low	Low	High	Ex-High
TLP (MSPT)	Ex-High	High	Ex-High	Ex-Low

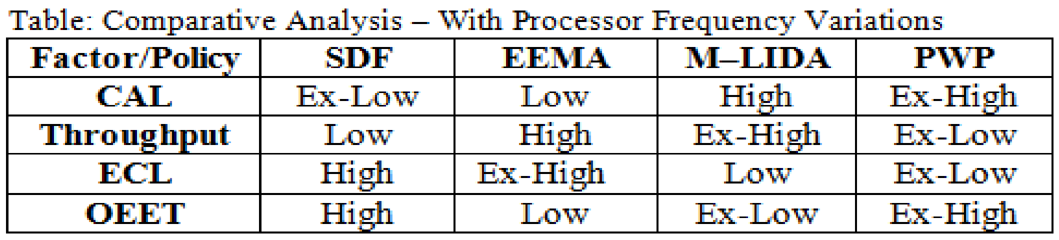


Table: Comparative Analysis – With Processor Frequency Variations

Factor/Policy	SDF	EEMA	M–LIDA	PWP
CAL	Ex-Low	Low	High	Ex-High
Throughput	Low	High	Ex-High	Ex-Low
ECL	High	Ex-High	Low	Ex-Low
OEET	High	Low	Ex-Low	Ex-High

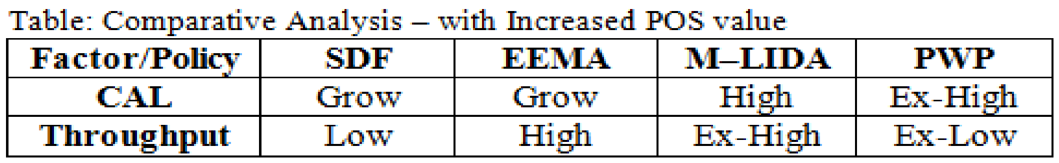


Table: Comparative Analysis – with Increased POS value

Factor/Policy	SDF	EEMA	M–LIDA	PWP
CAL	Grow	Grow	High	Ex-High
Throughput	Low	High	Ex-High	Ex-Low

5.2 Future Work

Future work will contain other dynamic policy methods to incorporate best processor managed space i.e. possibly more no. of parallel jobs with increased throughput as well as less excessive processor allocation. Usually the parallel jobs itself describes its processing demands i.e. no. of processor required. However this factor doesn't remains constant throughout the jobs execution life cycle. So there should be a mechanism to detect job's processing deeds by the scheduler itself, one approach to automatic detection of the job demand is based upon the DAT (Directed Acyclic Tree). Each job processes its execution with in a no. of phases. Ultimately a phase driven behaviour with in a parallel job execution model Parallel job driven model encompasses no. of thread to work out within a cooperative environment. So initially, during the earlier phases of job execution life cycle the no. of child threads encompasses are very less. As the phases completes towards their final attempt the no. of threads within each phase will vary so demand will be automatically identified by the scheduler. Further the cluster grid computing can be employed to evaluate performance of parallel algorithms. Network Architecture will behave like a parallel cluster for high data and computation intensive work, future study may incorporate cluster base experimentation. MPI based parallel interfaces are included for communicating control messages [10] [11]. PVM may be employed for parallel control constructs to for grid computing, cluster interconnection may designed for a particular set of application however load may be balanced among clusters for synchronizing .

[1] Sudha Srinivasan Savitha Krishnamoorthy P. Sadayappan, The Ohio State University, Columbus.A Robust Scheduling Strategy for Moldable Scheduling of Parallel Jobs. Proceedings of the IEEE International Conference on Cluster Computing

[2] Walfredo Cirne, Francine Berman. A Model for Moldable Supercomputer Jobs 2001 ACM Transactions, IPDS International Parallel & Distributed Processing Symposium. PP-59.

[3] Klaus Jansen Institut for Informatik, University zu Kiel. A (3/2+ε). Approximation Algorithm for Scheduling. SPAA'12, June , 2012, Pittsburgh, Pennsylvania, USA.

[4] Steven Hofmeyr, Costin Iancu Filip,Blagojevic. Load Balancing on Speed. Lawrence Berkeley National Laboratory. PPoPP'10, January , 2010, Bangalore, India.ACM

[5] Yves Caniou,Ghislain Charrier, Frederic Desprez. Evaluation of Reallocation Heuristics for Moldable Tasks in, 9th Australian Symposium on Parallel and Distributed Computing (AusPDC 2011), Perth, Australia.

[6] J.T. Moscicki, M. Lamannaa, M. Bubak, P.M.A. Sloot Processing moldable tasks on the grid: Late job binding with lightweight user-level overlay, www.elsevier.com/locate/fgcs, Feb 2011.

[7] Vignesh T. Ravi, Michela Becchi, Wei Jiang, Gagan Agrawal, Srimat Chakradhar Scheduling concurrent applications on a cluster of CPU-GPU nodes, July 2013.

[8] M. Etinski, J. Corbalan, J. Labarta, M. Valero.Parallel job scheduling for power constrained HPC systems, Sept 2012.

[9] Walfredo Cirne Francine Berman Using Moldability to Improve the Performance of Supercomputer Jobs. 2002 Elsevier Science.

[10] Erik Saule, Doruk Bozdaga, Ümit V. Çatalyürek Optimizing the stretch of independent tasks on a cluster: From sequential tasks to moldable tasks. January 2012.

[11] Gladys Utrera, Julita Corbalán, Jesús Labarta. Implementing Malleability on MPI Jobs 13th International Conference on Parallel Architecture and Compilation Techniques (PACT'04).

[12] Allen B. Downey, A parallel workload model and its implications for processor allocation, University of California at Berkeley and San Diego Supercomputer Center (1997).

[13] Ligang He, Stephen A. Jarvis, Daniel P. Spooner, Xinuo Chen and Graham R. Nudd Dynamic Scheduling of Parallel Jobs with QoS Demands in Multiclusters and Grids IEEE/ACM International Workshop on Grid Computing (GRID'04)

[14] Amit Chhabra, Gurvinder Singh and Gaurav Kumar, Simulated performance analysis of multiprocessor Dynamic space sharing Scheduling policies, International Journal of Computer Science and Network Security [2009].