International Journal of Computer Theory and Engineering, Vol. 1, No. 5, December, 2009
1793-8201

# Process Grain Size Based Scheduling of Parallel Jobs with Agile Algorithm

S.V.Sudha, K.Thanushkodi ,Kalignar Karunanidhi Institute of Technology,Coimbatore

***Abstract*—A good way of characterizing a parallel system is to consider the synchronization granularity or frequency of synchronization between processes in a system. The scientific applications of the parallel system consist of multiple processes running on different processors that communicate frequently. The performance evaluation of such systems mainly depends on how the processes are co scheduled. If the processes are not co scheduled properly, then the system will lead to severe performance penalties. The various co scheduling techniques available are First Come First Served, Gang Scheduling and Flexible Co Scheduling. First Come First Served and Gang Scheduling suffer from internal and external fragmentation. Flexible Co Scheduling saturates at heavy loads. The paper focuses on a new co scheduling algorithm, which concentrates on a detailed classification of the synchronization granularity, and the new algorithm gives better results under heavy loads.**

***Index Terms*— First Come First Served, Flexible Co Scheduling, Gang Scheduling, Parallel Job Scheduling, Performance Metrics.**

## I. INTRODUCTION

Scheduling parallel jobs for execution is similar to bin packing. Each job needs a certain number of processors for a certain time and the scheduler has to pack these jobs together so that most of the processors will be utilized most of the time. In job scheduling, Synchronization overhead could turn to be key issue for utilizations of the processors[1]. If Scheduling does not carefully address the synchronization overhead, the utilization of each processor in a parallel system can end up comparatively lower than a single processor system.[13]

The domain, we will use is the scheduling of parallel jobs for execution on a parallel system, Such Scheduling is typically done by partitioning the machine's processors and running a job on each partition. This is similar to packing in two dimensions [4][5]. Regarding one dimension as representing processors and the other as representing time. A Parallel job is a rectangle, representing the use of a certain number of processors for certain duration of time. The scheduler has to pack these rectangles as tightly as possible within the space provided by the available resources.[12][14] The sizes of the rectangles are known as each submitted job comes with a specification of how many processors to use, and an estimate of how long it will run. Due to the synchronization between processes in a job, the jobs do not pack perfectly; therefore holes are left in the schedule. If the processes are not co scheduled properly, it will harm the performance of the parallel algorithm.

The co scheduling algorithms available are First Come First Serve, Gang Scheduling and Flexible Co scheduling .The main drawback of First Come First Serve is the central queue occupies a region of memory that must be accessed in a manner that enforces mutual exclusion. [10][11].Thus it may become a bottleneck if many processors look for work at the same time. If all threads are treated as a common pool of threads, it is unlikely that all of the threads of a program will gain access to processors at the same time.

If a high degree of coordination is required between the threads of a program, the process switches involves many seriously compromise performance [2][3]. Gang scheduling requires that the schedule of communicating processes be precomputed which complicated the co scheduling of client server applications and requires pessimistic assumption about which processes communicate with one another. Flexible Co scheduling saturates at higher loads.[1]

In this paper, we show that it is possible to increase the resource utilization with various synchronization granularities between the processes. We introduce a new methodology called Agile Scheduling which classifies a detailed granularity of processes and shows better results than the above mentioned ones.

## II. EVALUATION METHODOLOGY

Before we present our results, we first need to describe our methodology. In this section, we begin by describing the characteristics of the workloads we use. Next we discuss about the performance metrics we adopt to measure the quality of service in the system

### *A. Agile Algorithm*

The Algorithm concentrates on detailed classification of the frequency of synchronization between processes in a system. The processes are classified as[1][5]

1) Fine Grain

Fine Grained Parallelism represents a much more complex use of parallelism .The processes communicate often and must be co scheduled effectively due to their demanding synchronization. [9]

2) Medium Grain

Medium Grain Parallelism represents enough synchronization between the processes and the scheduling algorithms should take care of the performance evaluation of

Manuscript received August 14, 2009.

S.V.Sudha ,working as Assistant Professor in the Department of Information Technology, Kalignar Karunanidhi Institute of Technology,Coimbatore 641 402 ,Tamil Nadu ,India ( e-mail: svsudha@rediffmail.com)

K.Thanushkodi ,Principal ,f Akshaya College of Engineering and Technology, Coimbatore -642 109,Tamil Nadu,India.

the system.

3) Coarse Grain

With Coarse Grain, there is synchronization among processes, but at a very gross level. This kind of situation is easily handles as a set of concurrent processes running on a multiprogrammed uniprocessor and can be supported on a multiprocessor with little or no change to the software.

4) Independent

With Independent parallelism, there is no explicit synchronization among processes . Each represents a separate, independent application or job.

### *B. Workload Characteristics*

The simulation studies were performed using the Agile Algorithm with workload logs available from Feitelson's Archive.[15][8]

1) Fine Grain Workload

The log considered for the experiment was The Los Alamos National Lab (LANL) Log. This log contains two years worth of accounting records produced by the DJM software running on the 1024 node CM-5 at Los Alamos. Total Number of jobs present in the Log is 2, 01,387 Jobs

The Log contains the following details 1) Job Id 2) Submit Time 3) Start Time and Date 4) End Date and Time.

2) Medium Grain Workload

The Log Considered for the experiment was LLNL Thunder Log. This log contains several months' worth of Accounting Records from a large Linux Cluster called Thunder installed at Lawrence Livermore. Total Number of jobs present in the Log is 1, 28,662 Jobs

The Log contains the following details 1) Job ID 2)User ID 3) Name 4) Job State 5)Start Time 6)End Time

3) Coarse Grain Workload

The Log considered for the experiment was The Lawrence Livermore National Lab (LLNL) T3D Log. This Log contains 4 months worth of Accounting Record at the Lawrence Livermore National Lab (LLNL). Total Number of jobs present in the Log is 21323 Jobs.

The Log contains the following details 1) Start Date 2) Start Time 3) Process ID 4) Partition ID.

4) Independent

The Log considered for the experiment was LPC Log. This Log contains 9 Months of Record .Total number of Jobs present in the Log is 2, 44,821 Jobs.

The Log Contains the following details 1) Job ID 2) Submit Time 3) Wait Time 4) Run Time.

### C. Performance Metrics

The synthetic workload generated Feitelson's archive are used as input to the simulation of various scheduling strategies. We monitor the following parameters the arrival time, start time, execution time; finish time etc .Different Scheduling algorithms have different properties and may favor one class of processes over another. In choosing which algorithm to use in a particular situation, we must consider the properties of the various algorithms. Many criteria have been suggested for scheduling algorithms. The criteria includes the following

Mean Utilization:

We want to keep the CPU as busy as possible. CPU Utilization may range from 0 to 100 percent. In a real system, it should range from 40 percent (for a lightly loaded system) to 90 percent (for a heavily loaded system).The mean utilization is the ratio of cpu busy time to the number of processors multiplied with Total time for execution.[1][6][7]

$$\text{Mean Utilization} = \frac{\Sigma \text{ CPU Busy Time}}{\text{Number of Processors} * \text{Total Time}} \quad (1)$$

Mean Response Time

In an interactive system, Turnaround time may not be the best criteria. Often, a process can produce some output fairly early, and can continue computing new results while previous results are being output to the user. Thus, another measure is the time from the submission of a request until the first response is produced. This measure is called response time

$$\text{Mean Response Time} = \frac{\Sigma \text{ Job Finish Time-Job Submit Time}}{\text{Number of Jobs}} \quad (2)$$

Mean Reaction Time

The mean job reaction time defined as the mean time interval between the submission and the start of the job.

$$\text{Mean Reaction Time} = \frac{\Sigma \text{ Job Start Time-Job Submit Time}}{\text{Number of Jobs}} \quad (3)$$

Mean Slowdown

Mean Slowdown is the sum of jobs response times divided by the job's execution times. This metric emerges as a solution to normalize the high variation of the jobs response time.

$$\text{Mean Slow down} = \frac{\Sigma \text{ Job Response Time/JobExecution time}}{\text{Number of Jobs}} \quad (4)$$

Turn Around Time

From the point of view of a particular process, the important criterion is how long it takes to execute that process. The interval from the time of submission of a process to the time of completion is the turn around time. Turn around time is the sum of periods spent waiting to get into memory, waiting in the ready queue, executing on the CPU and doing I/O.

Waiting Time

The scheduling algorithm does not affect the amount of time during which a process executes or does I/O;it affects only the amount of time that a process spends waiting in the ready queue. Waiting time is the sum of the periods spent waiting in the ready queue.

## III. SCHEDULING STRATEGY

### *A. Fine Grain Application Algorithm*

for (all jobs in a queue)
sort the jobs in accordance with the submit time
divide the total number of jobs in to 1000 slots
each job is given a time quantum so that a strict global round robin is followed.
while (slots available)
while (number of jobs available in the slot)

Assign the jobs in the Scheduling Matrix. (ie) each job in a time slice.

*B. Medium Grain Application Algorithm*

for (all jobs in a queue)
sort the jobs in accordance with the submit time
divide the total number of jobs in to 1000 slots
each Job is given a time quantum so that a strict Global Round Robin is followed.
while(slots available)
Scheduling flag is ON
Scheduling Count is 0.
while(number of jobs available in the slot)
if scheduling flag
Primary jobs =5 jobs from the slot
Assign the jobs in the primary slots of the scheduling matrix.
Secondary slots =ideal processors not used by the primary jobs.
Scheduling flag is off.
Scheduling count is incremented.
else
Secondary jobs =next 5 jobs in the slot.
if secondary slots available
assign secondary jobs in the secondary slots.
Scheduling flag is ON
Scheduling count is incremented.
if secondary jobs still available
Schedule the first time slice jobs to the processor and make free to assign the remaining jobs.
if scheduling count is 2
Schedule the jobs to the processor
Scheduling count is 0.
Schedule the jobs to the Processor.

*C. Coarse Grain Application Algorithm*

for (all jobs in a queue)
sort the jobs in accordance with the submit time
divide the total number of jobs in to 1000 slots.
divide the 1000 slots in to two and name as slot head i and ii
each slot head contains 500 slots in each slot head.
each job is given a time quantum so that a strict Global Round Robin is followed.
while (number of slots available in the slot head i and ii)
scheduling flag is on
scheduling count is 0.
while (number of jobs available in the slots of the slot head
I and II)
if scheduling flag
primary jobs =5 jobs from the slots of the slot head I
Assign the jobs in the primary slots of the scheduling matrix.
Secondary slots =ideal processors not used by the primary jobs.
Scheduling flag is off.
Scheduling count is incremented
**e**lse
Secondary jobs = 5 jobs from the slot of the slot head II
If secondary slots available assign secondary jobs in the secondary slots.
Scheduling flag is ON
Scheduling count is incremented
if secondary jobs still available
Schedule the first time slice jobs to the processor and make free to assign the remaining jobs
if scheduling count is 2
Schedule the jobs to the processor
Scheduling count is 0.

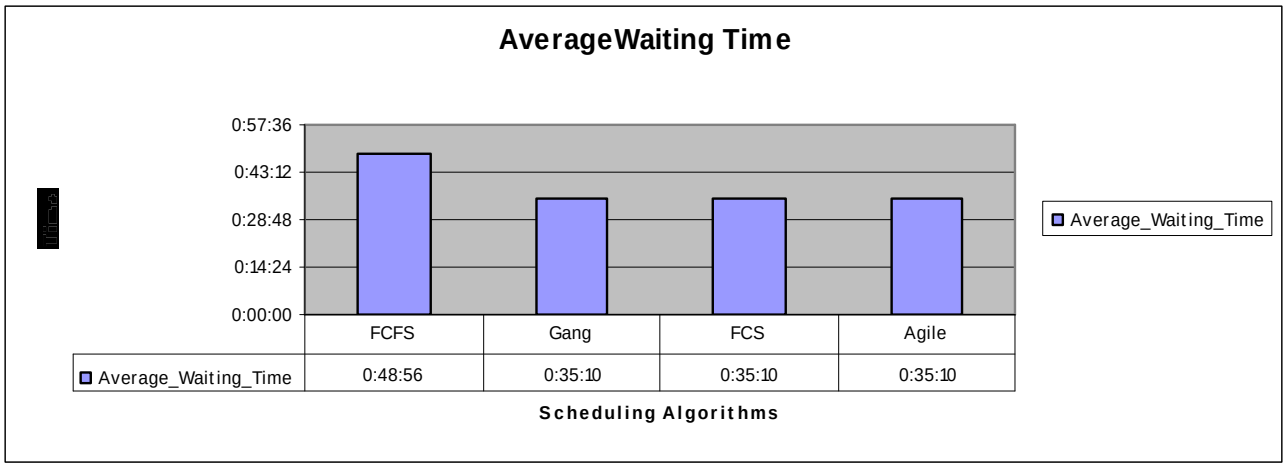
*D. Independent*

Any type of Grain Algorithm can be applied

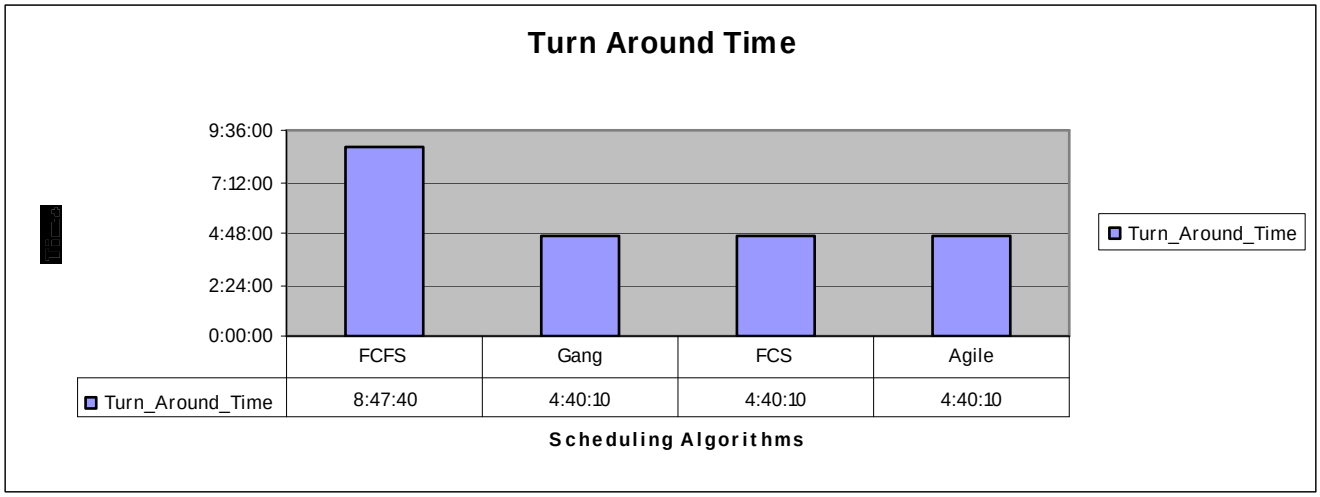
## IV. Experimental Results

In this section, we present and analyze the performance of Agile Algorithm. First, for each metric, we present the results by simulation. All simulators are written in Java.

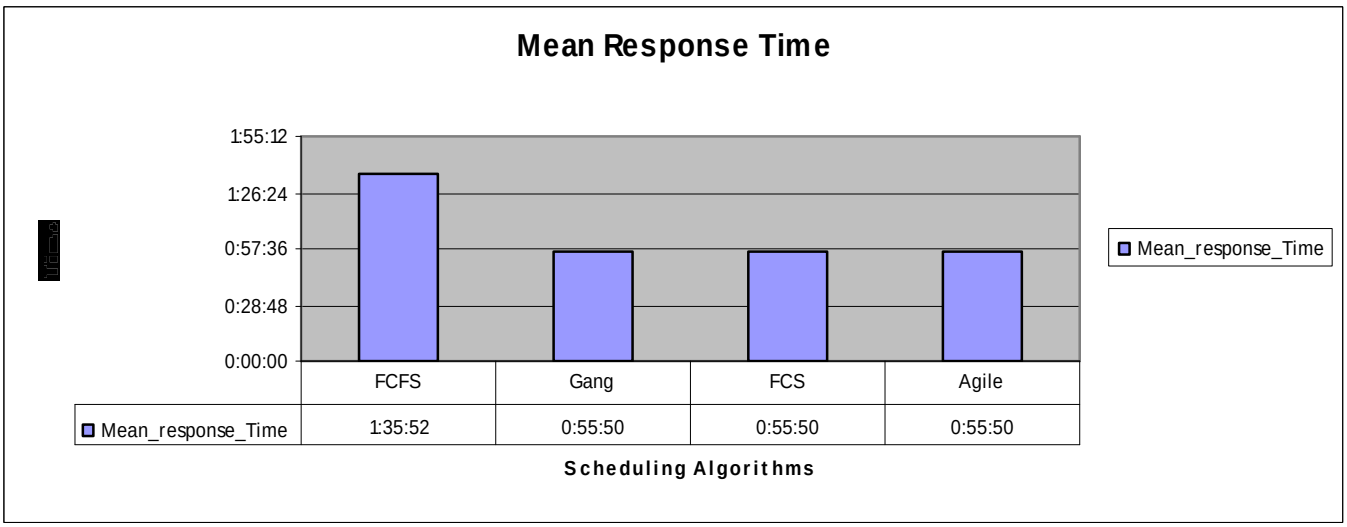
*A. Fine Grain Workloads*



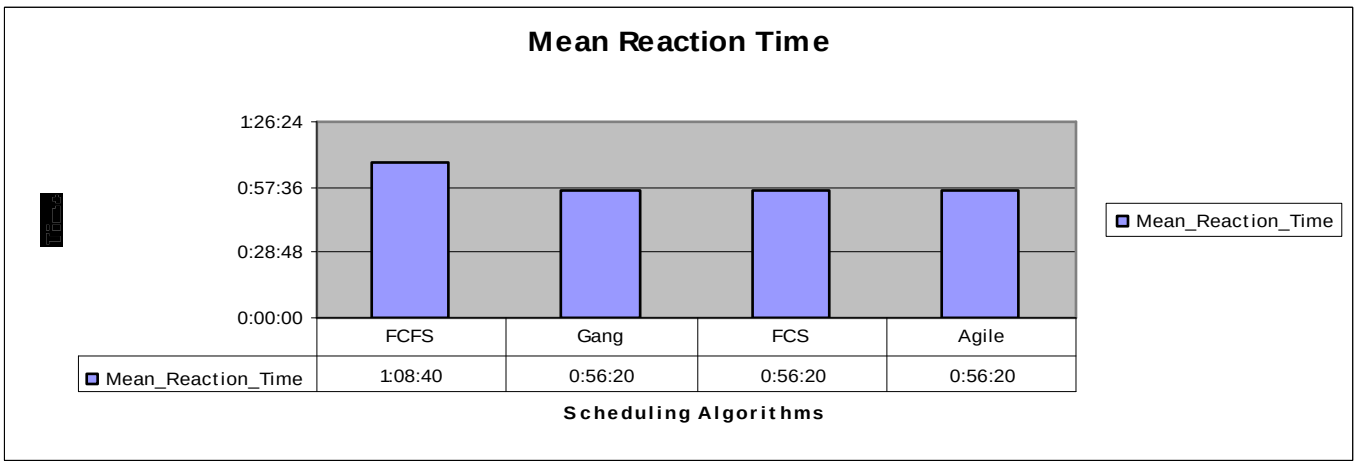

**(a)**


Turn Around Time
9:36:00
7:12:00
4:48:00
2:24:00
0:00:00
FCFS
Gang
FCS
Agile
Turn_Around_Time
8:47:40
4:40:10
4:40:10
4:40:10
Scheduling Algorithms
Turn_Around_Time

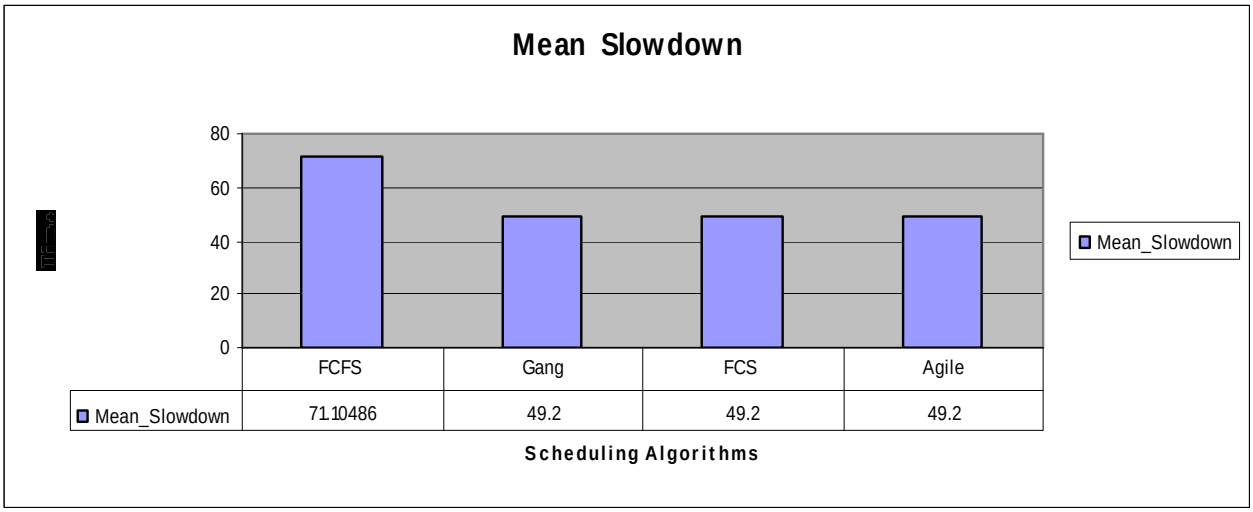
**(b)**


Mean Response Time
1:55:12
1:26:24
0:57:36
0:28:48
0:00:00
FCFS
Gang
FCS
Agile
Mean_response_Time
1:35:52
0:55:50
0:55:50
0:55:50
Scheduling Algorithms
Mean_response_Time

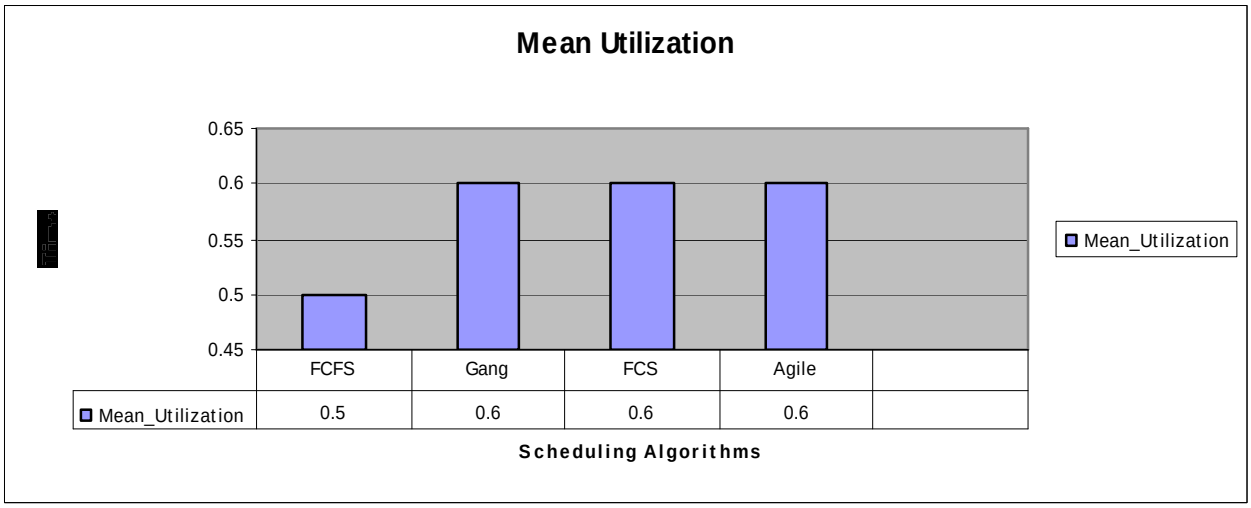
**(c)**


Mean Reaction Time
1:26:24
0:57:36
0:28:48
0:00:00
FCFS
Gang
FCS
Agile
Mean_Reaction_Time
1:08:40
0:56:20
0:56:20
0:56:20
Scheduling Algorithms
Mean_Reaction_Time

**(d)**



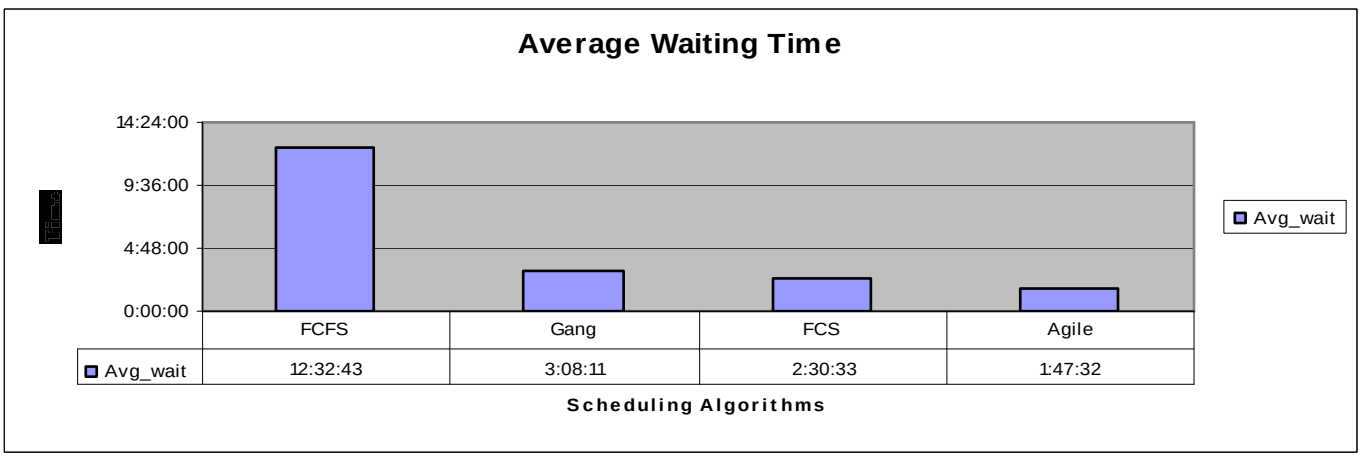

**(e)**



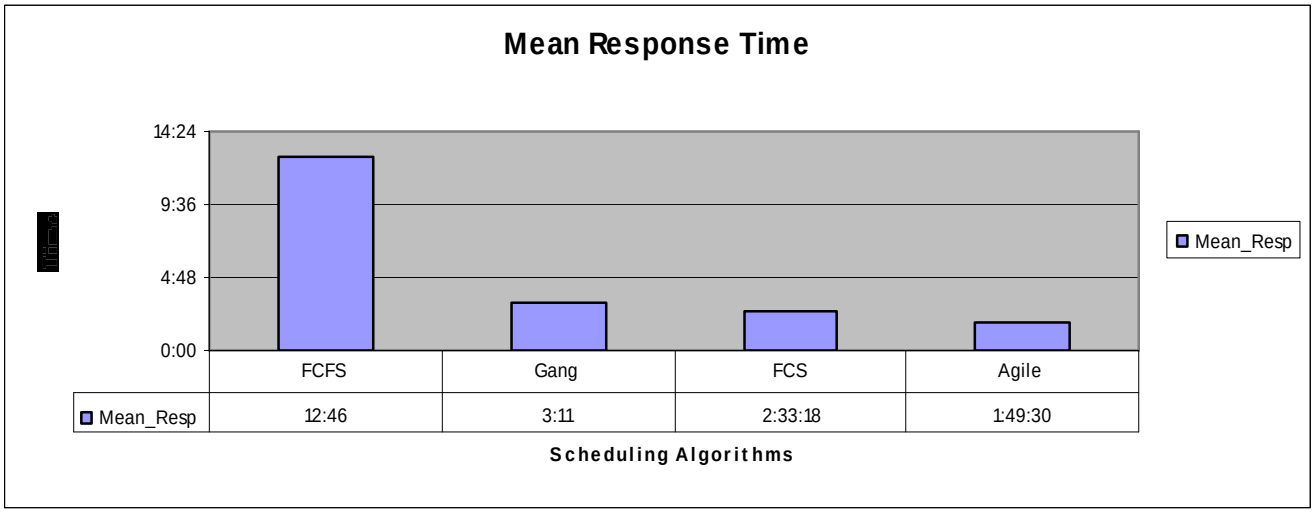

**(f)**

Fig 1. Performance Evaluation of Agile Algorithm for Fine Grain Application with First Come First Served, Gang Scheduling, Flexible Co scheduling. (a) Average Waiting Time (b) Turn Around Time (c) Mean Response Time (d) Mean Reaction Time (e) Mean Slowdown (f) Mean Utilization

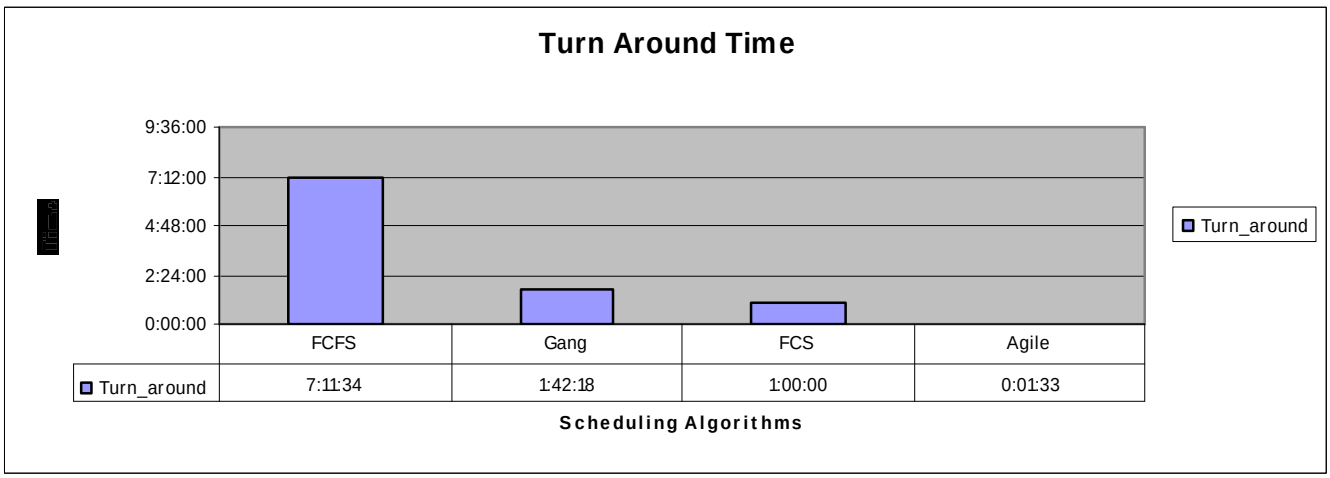
*B. Medium Grain Workloads*



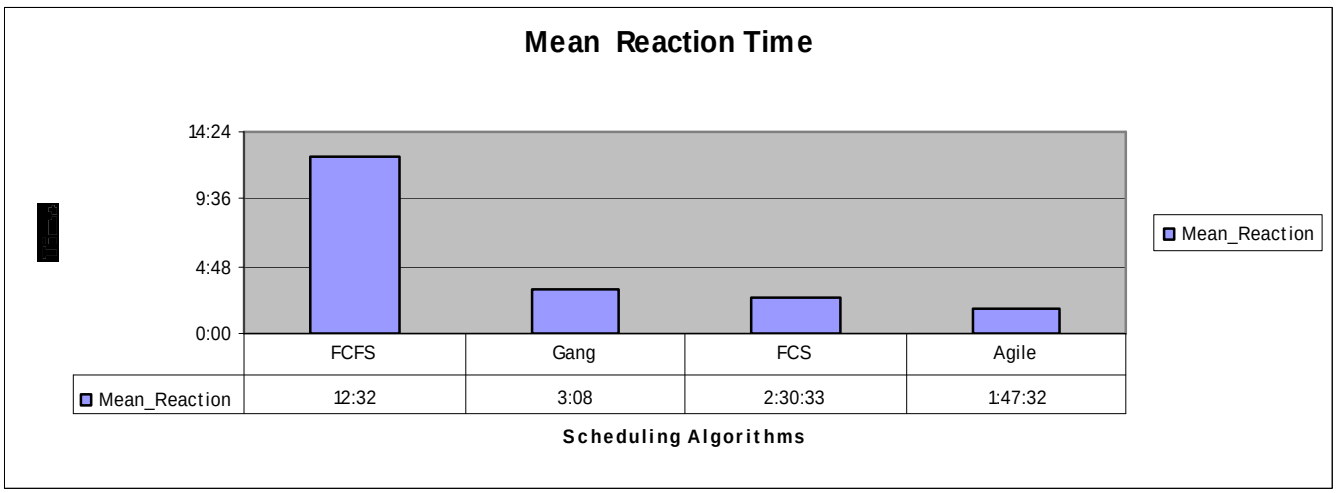

**(a)**


Mean Response Time
14:24
9:36
4:48
0:00
FCFS
Gang
FCS
Agile
Mean_Resp
12:46
3:11
2:33:18
1:49:30
Mean_Resp
Scheduling Algorithms

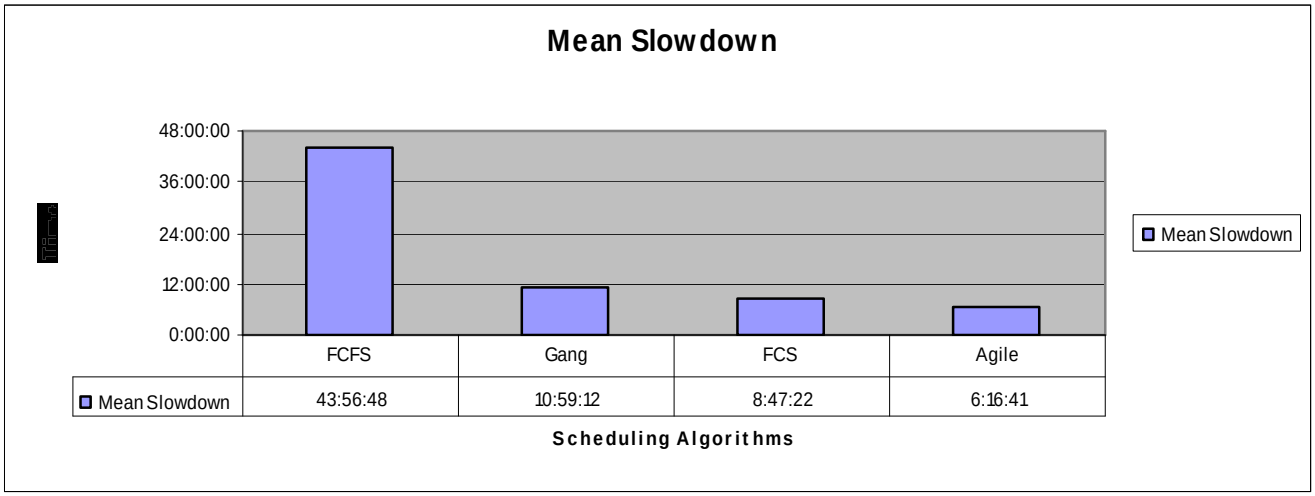
**(b)**


Turn Around Time
9:36:00
7:12:00
4:48:00
2:24:00
0:00:00
FCFS
Gang
FCS
Agile
Turn_around
7:11:34
1:42:18
1:00:00
0:01:33
Turn_around
Scheduling Algorithms

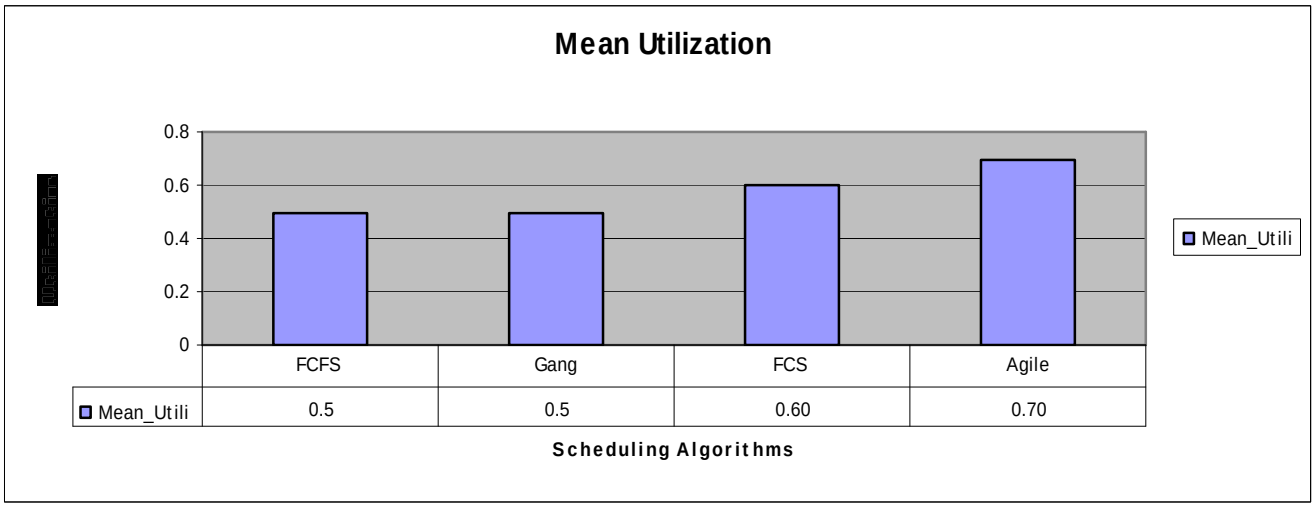
**(c )**


Mean Reaction Time
14:24
9:36
4:48
0:00
FCFS
Gang
FCS
Agile
Mean_Reaction
12:32
3:08
2:30:33
1:47:32
Mean_Reaction
Scheduling Algorithms

**(d)**



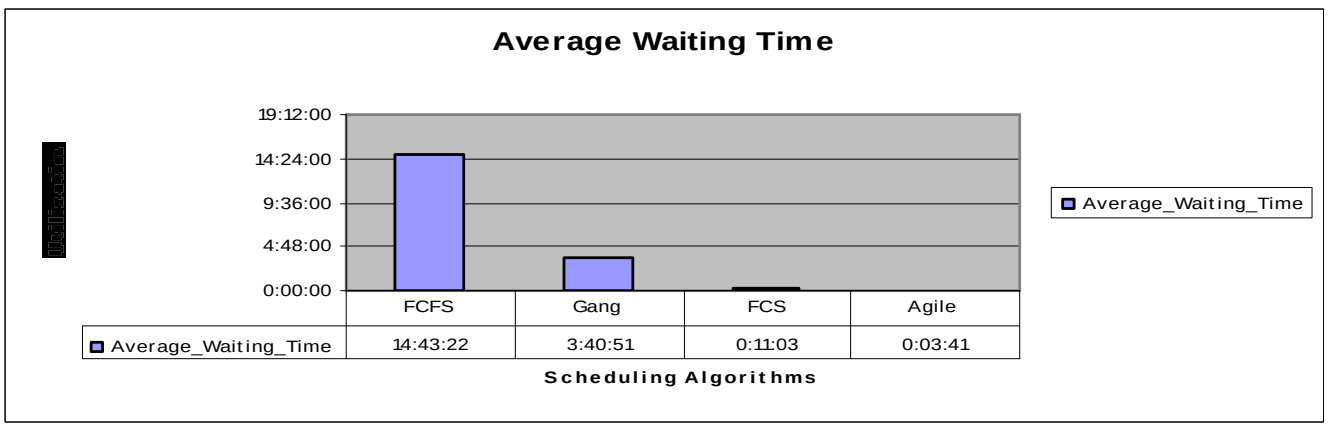

**(e)**



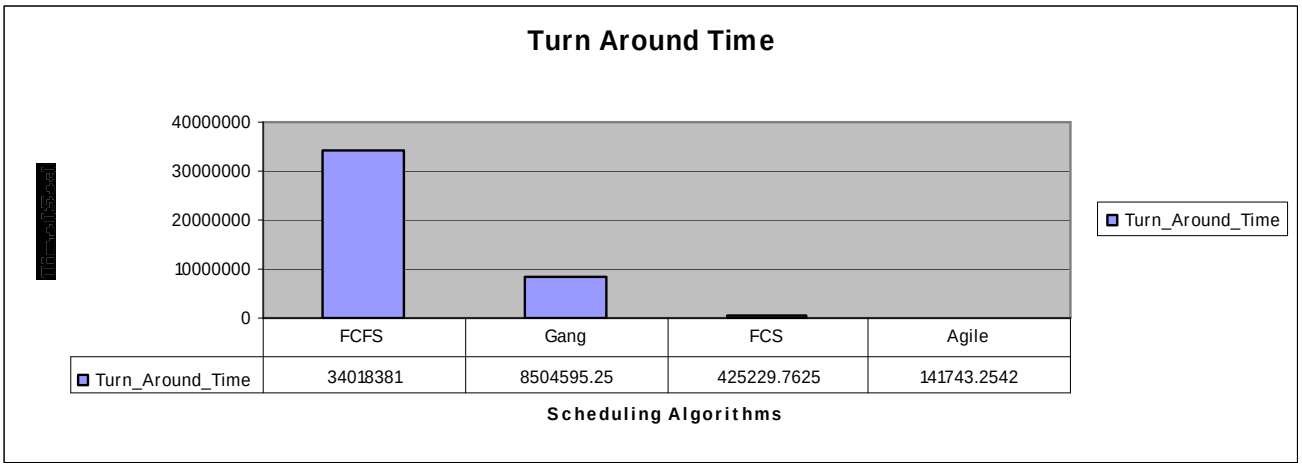

**(f)**

Fig 2. Performance Evaluation of Agile Algorithm for Medium Grain Application with First Come First Served, Gang Scheduling, Flexible Co scheduling. (a) Average Waiting Time (b) Mean Response Time(c) Turn Around Time (d) Mean Reaction Time (e) Mean Slowdown (f) Mean Utilization.

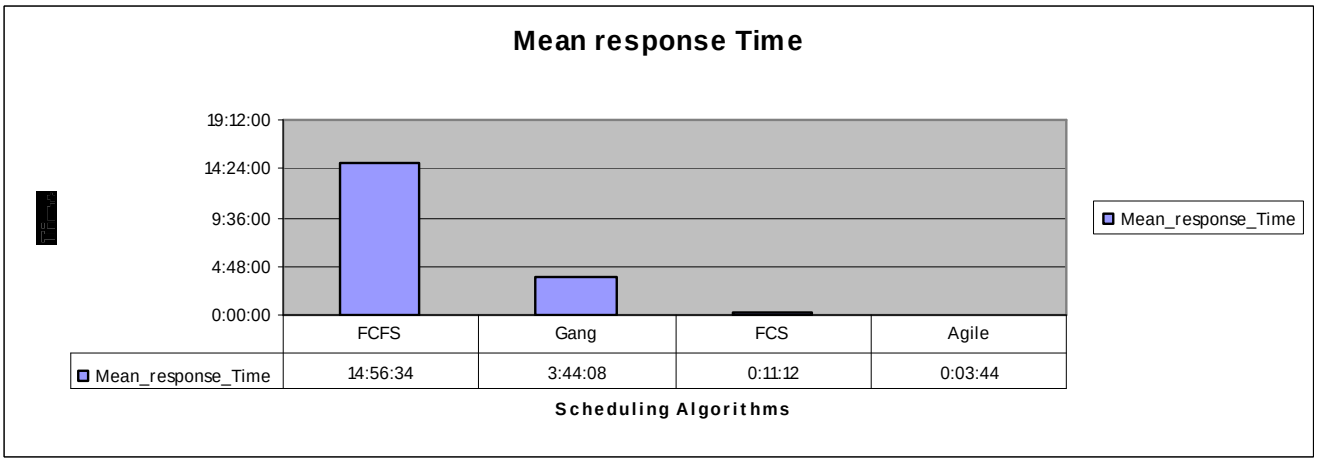
*C. Coarse Grain Workloads*



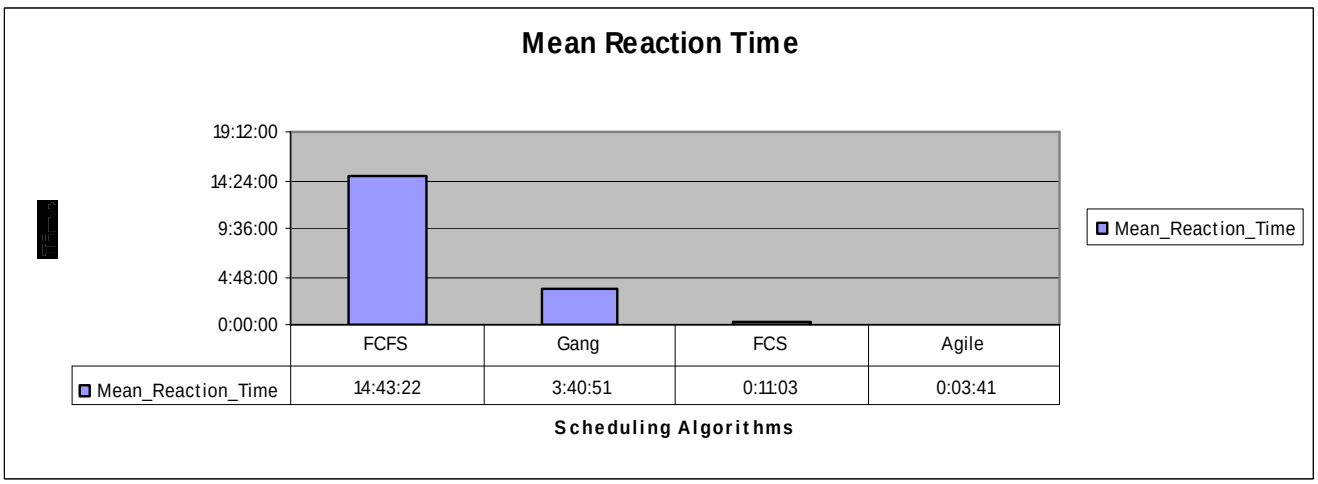

**(a)**


Turn Around Time
40000000
30000000
20000000
10000000
0
FCFS
Gang
FCS
Agile
Turn_Around_Time
34018381
8504595.25
425229.7625
141743.2542
Scheduling Algorithms
Turn_Around_Time

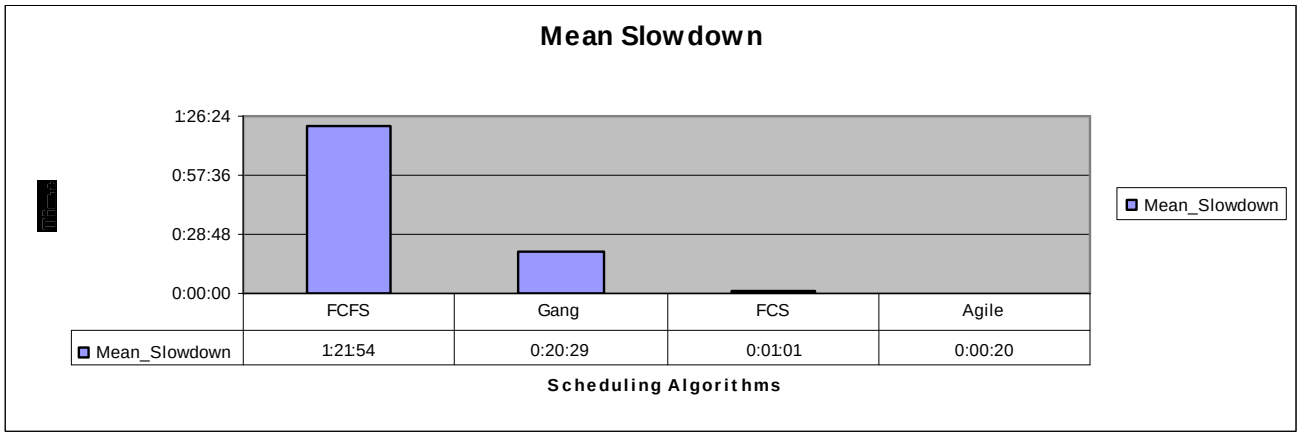
**(b)**


Mean response Time
19:12:00
14:24:00
9:36:00
4:48:00
0:00:00
FCFS
Gang
FCS
Agile
Mean_response_Time
14:56:34
3:44:08
0:11:12
0:03:44
Scheduling Algorithms
Mean_response_Time

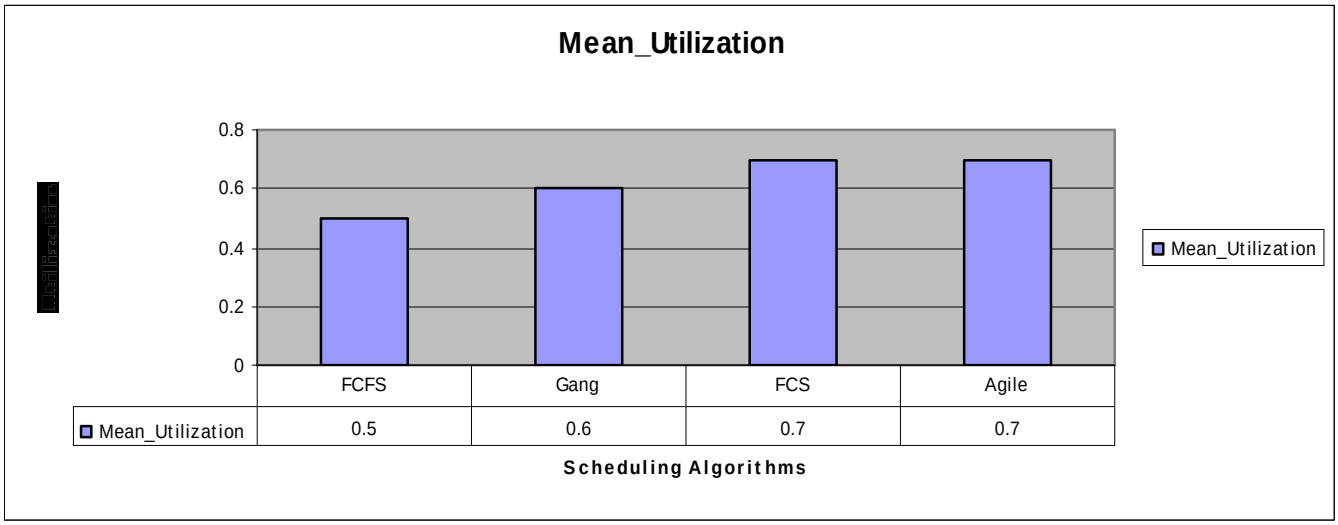
**(c )**


Mean Reaction Time
19:12:00
14:24:00
9:36:00
4:48:00
0:00:00
FCFS
Gang
FCS
Agile
Mean_Reaction_Time
14:43:22
3:40:51
0:11:03
0:03:41
Scheduling Algorithms
Mean_Reaction_Time

**(d)**



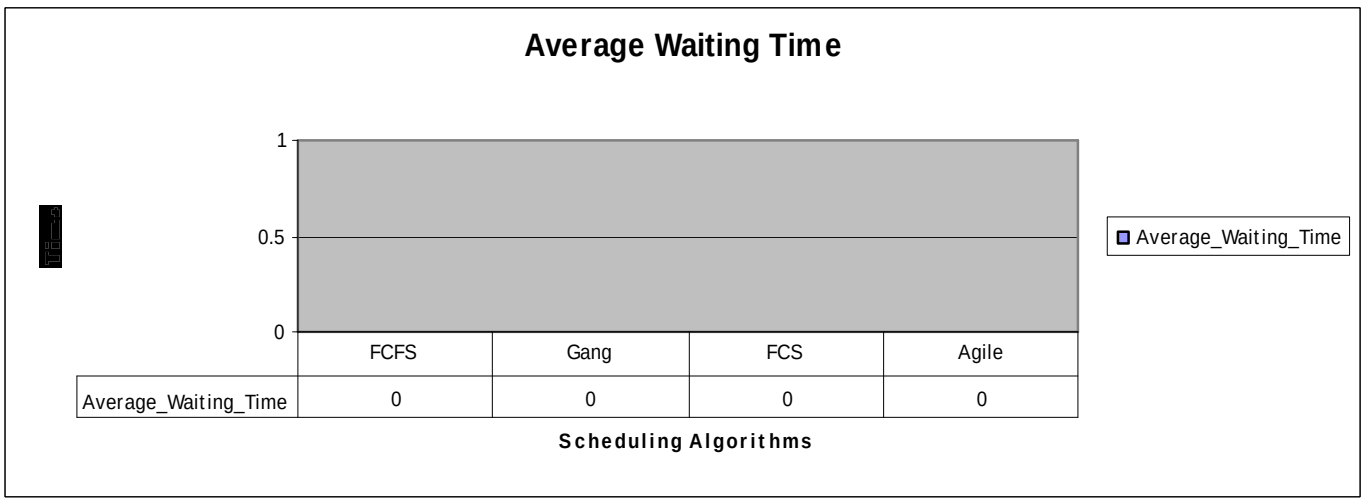

**(e)**



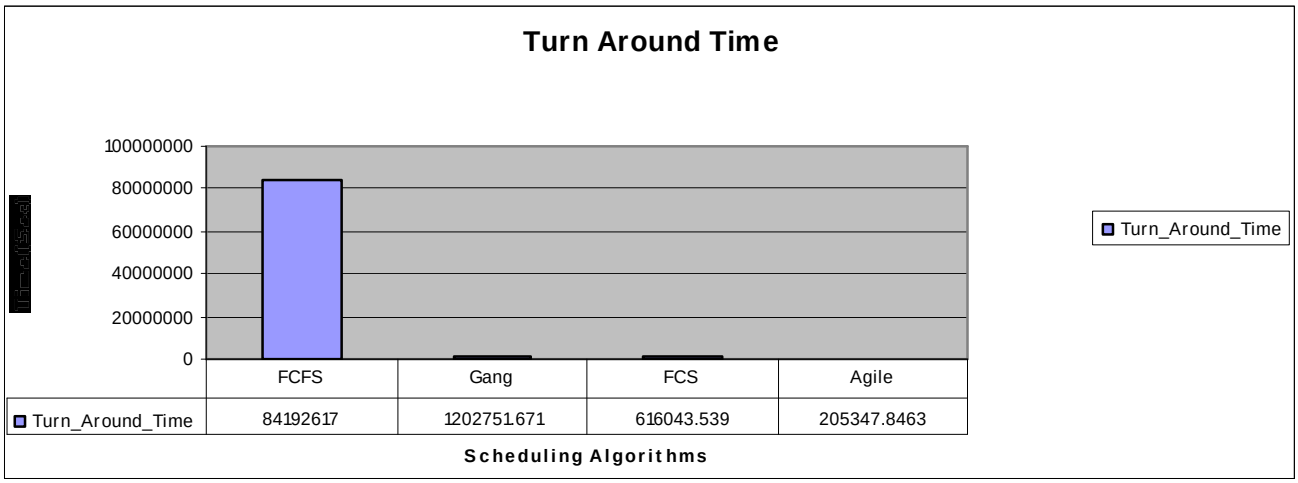

**(f)**

Fig 3 Performance Evaluation of Agile Algorithm for Coarse Grain Application with First Come First Served, Gang Scheduling, Flexible Co scheduling. (a) Average Waiting Time (b) Turn Around Time(c) Mean Response Time (d) Mean Reaction Time (e) Mean Slowdown (f) Mean Utilization

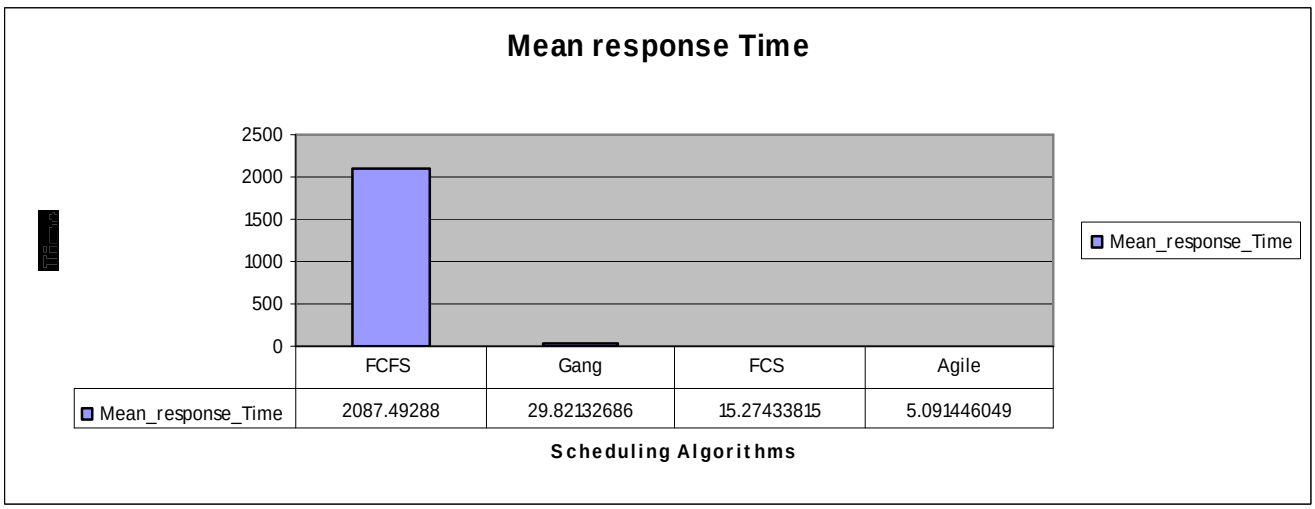
*D. Independent*



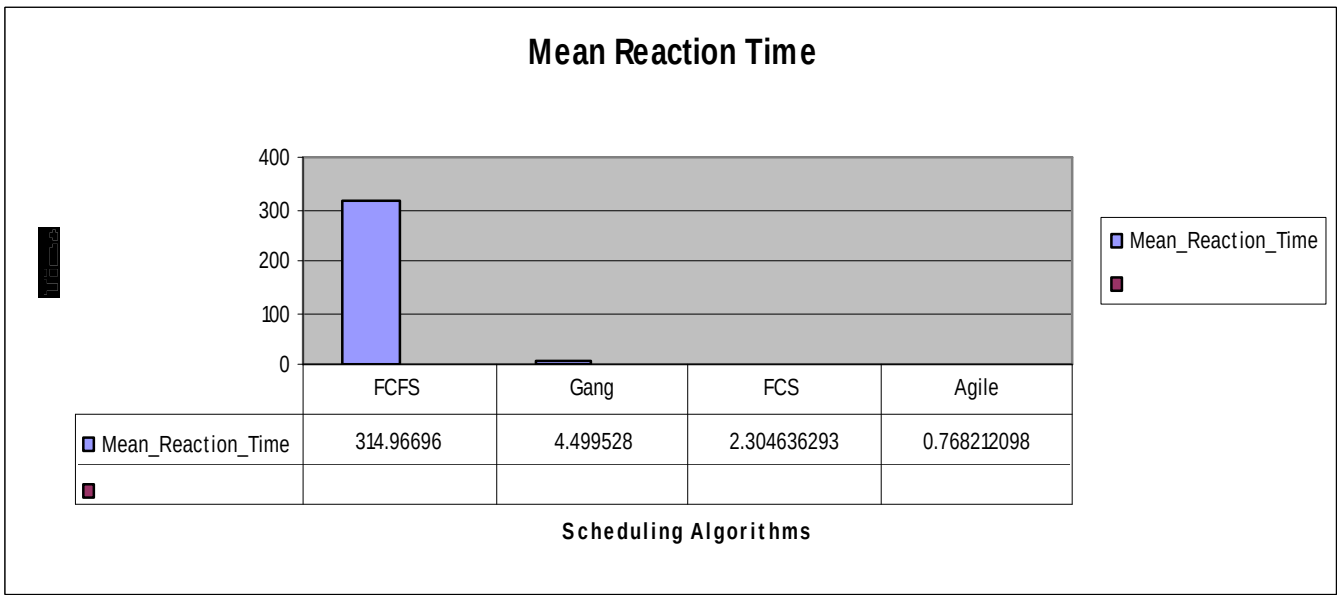

**(a)**


Turn Around Time
100000000
80000000
60000000
40000000
20000000
0
FCFS
Gang
FCS
Agile
Turn_Around_Time
84192617
1202751671
616043.539
205347.8463
Scheduling Algorithms
Turn_Around_Time

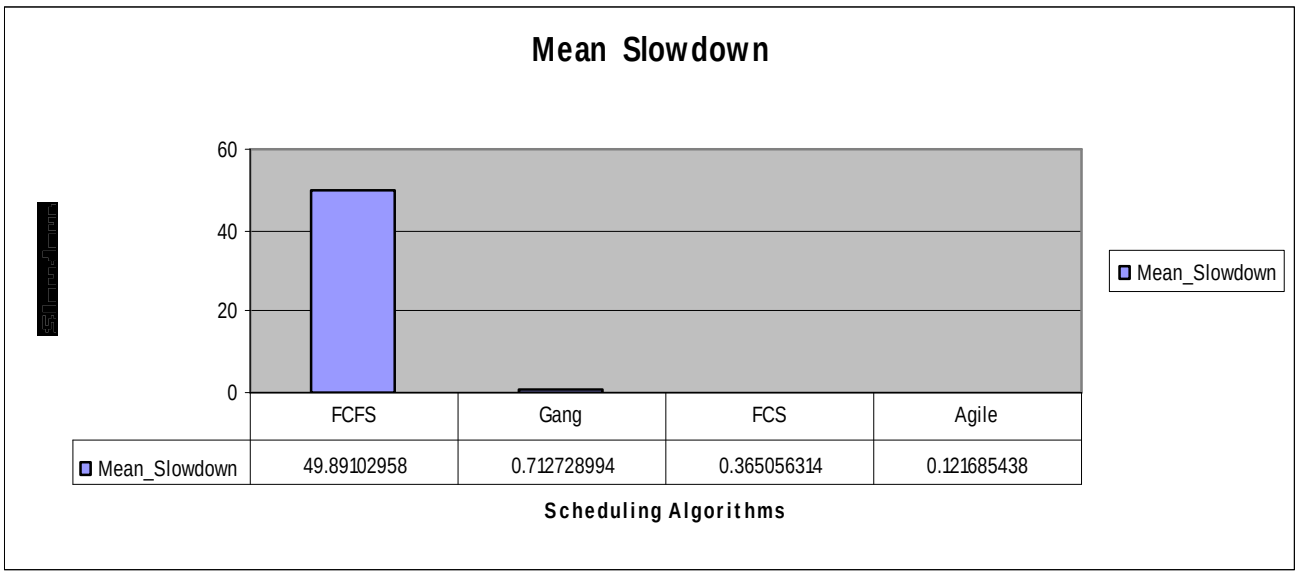
**(b)**


Mean response Time
2500
2000
1500
1000
500
0
FCFS
Gang
FCS
Agile
Mean_response_Time
2087.49288
29.82132686
15.27433815
5.091446049
Scheduling Algorithms
Mean_response_Time

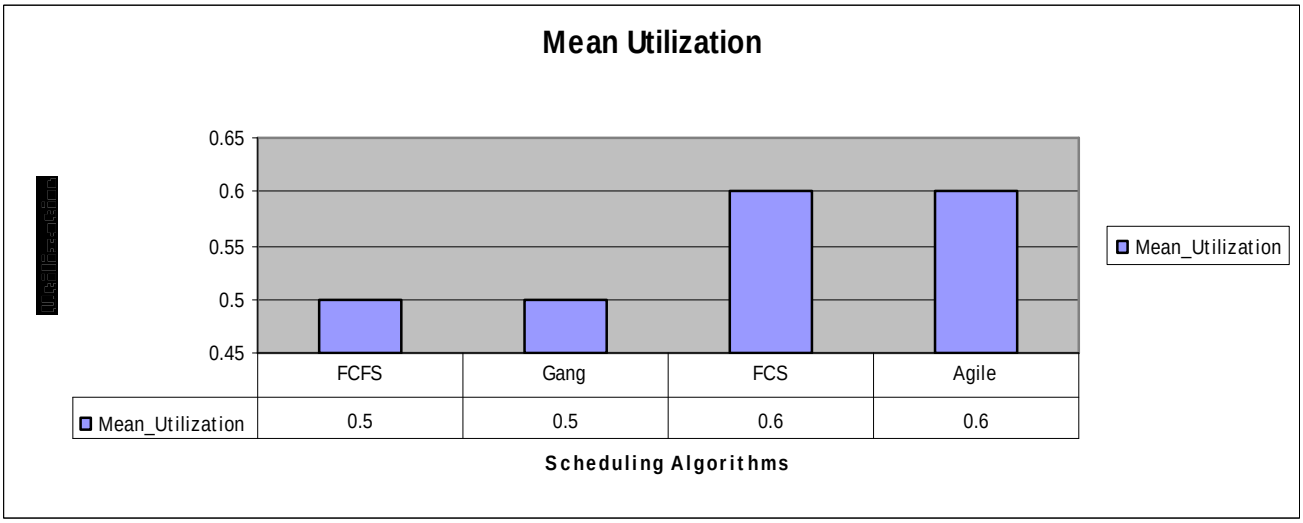
**(c)**


Mean Reaction Time
400
300
200
100
0
FCFS
Gang
FCS
Agile
Mean_Reaction_Time
314.96696
4.499528
2.304636293
0.768212098
Scheduling Algorithms
Mean_Reaction_Time

**(d)**




**(e)**




**(f)**

Fig 4. Performance Evaluation of Agile Algorithm for Independent Application with First Come First Served, Gang Scheduling, Flexible Co scheduling. (a) Average Waiting Time (b) Turn Around Time(c) Mean Response Time (d) Mean Reaction Time (e) Mean Slowdown (f) Mean Utilization

The Agile Algorithm concentrated on the detailed classification of the granularity of the processes and schedules jobs on the grain size. The results shows that the algorithm behaves as same as the gang scheduling for the fine synchronization of the jobs as shown in the Fig 1.For the Medium grain, coarse grain and Independent applications as shown in the figures 2,3, and 4 the algorithm gives better results for the parameters we have considered when compared to the other scheduling algorithms.

## V. Results & Discussions

For the Log LANL i.e. for the fine grain application, the overall running time with the First Come Served Algorithm was 2 hours, 31 minutes and 6 seconds. The overall running time with the Gang Scheduling was 17 minutes and 9 seconds. Flexible co scheduling and the Agile Algorithm give the same figure as Gang Scheduling.

For the Log LLNL i.e. for the medium grain application, the overall running time with the First Come Served Algorithm was 7 hours, 11 minutes and 34 seconds. The overall running time with the Gang Scheduling was 32 minutes and 27 seconds. The overall running time for the Flexible co scheduling was 16 minutes and 33 seconds and for the Agile Algorithm it was 1 minute and 33 seconds.

For the Log LLNL T3D i.e. for the coarse grain application, the overall running time with the First Come Served Algorithm was 2 hours, 21 minutes and 7 seconds. The overall running time with the Gang Scheduling was 17 minutes and 9 seconds. The overall running time for the Flexible co scheduling was 5 minutes and 13 seconds and for the Agile Algorithm it was 35 seconds.

For the Log LPC Log i.e. for the Independent grain application, the overall running time with the First Come Served Algorithm was 5 hours, 56 minutes and 2 seconds. The overall running time with the Gang Scheduling was 1 minutes and 10 seconds. The overall running time for the Flexible co scheduling was 40 seconds and for the Agile

Algorithm it was 16 seconds.

All the comparisons are clearly analyzed and are shown in the figures 1,2,3,4. Equations 1, 2,3 and 4 are being used for the comparisons.

## VI. Conclusions

We present a new Scheduling methodology Agile Algorithm for different Grain Applications. The Algorithm concentrates mainly on the frequency of synchronization between the processes of the application and the performance is improved and the agile algorithm is compared with First Come First Served, Gang Scheduling and Flexible Co scheduling. The real time workload for various grains is considered for the calculation.

Our Algorithm has overcome the traditional way in parallel job scheduling algorithms that the specialization for specific types of workloads, which results in poor performance when the workload characteristics do not fit the model for which they were designed. In all the scenarios, Agile Algorithm performs equally well or better than the other algorithm in terms of Turn around time; Average waiting time, mean response time, mean reaction time, mean slowdown and utilization

## References

[1] Eitan Frachtenberg, Dror G.Feitalson. Fabrizio Petrini,Juan Fernandez,"Adaptive Parallel Job Scheduling with Flexible Coscheduling",*IEEE Trans.Parallel and Distributed Systems,*Vol 16,No 11, pp.1066-1077,November 2005.

[2] Dan Tsafrir, Yoav Etsion, Dror G.Feitelson, " Backfilling Using System –Generated Predictions Rather than User Runtime Estimates", IEEE Transaction on Parallel and Distributed Systems , Vol 18, No 6, pp 789-803,June 2007.

[3] Avi Nissimov And Dror G.Feitelson, " Probabilistic Backfilling", 13th Workshop on Job Scheduling Strategies for Parallel Processing In Conjuction with 21st ACM International Conference on Super Computing ,pp 102-115,June 2007.

[4] C.Anglano , "A Comparative Evaluation of Implicit Coscheduling Strategies for Network of Workstations,"Proc. Ninth Int'l Symp. High Performance Distributed Computing, pp 221-228,August 2000.

[5] C.D.Antonopoulos, D.S.Nikolopoulos and T.S.Papatheodorou, "Informing Algorithms for Efficient Scheduling of Synchronizing Threads on Multiprogrammed SMPs",Proc Int'l Conf Parallel processing ,pp 123-130 ,Sept 2001.

[6] E.Franchtenberg,D.G.Feitelson ,J.Fernandez-Peinador and F.Petrini , "Parallel Job Scheduling under Dynamic Workloads ",Proc.Ninth Workshop Job Scheduling Strategies for Parallel Processing,pp 208-227,June 2003.

[7] U.Lublin and D.G.Feitelson , "The Workload of Super Computers"Modeling the Characteristics of Rigid Jobs" ,J.Parallel and Distributed Computing ,vol 63,no 11,pp 1105-1122 ,Nov 2003.

[8] R.Kettimuthu,V.Subramani, .Srinivasan ,T.B.Gopalasamy,D.K.Pand a and P.Sadayappa, "Selective preemption Strategies for Parallel Job Scheduling ",Proc Int'l Conf Parallel Processing ,pp 55-71,August 2002.

[9] [9] W.Lee ,M.Frank,V.Lee ,K.Mackenzie and L.Rudolph, Implication of I/O for Gang Scheduled Workloads," Job Scheduling Strategies for Parallel Processing ,pp 215-237, 1997.

[10] D.Kerbyson,H.Alme,A.Hoisie ,F.Petrini ,H.Waserman and Miggints ,. "Predictive Performance and Scalability Modeling of a Large –Scale SMP Clusters", Super Computing ,ACM/IEEE 2001 Conference,Volume ,issue 10-16 ,pp 30-39,Nov 2001.

[11] E.Frachtenberg, F.Petrini, J.Fernandez, S.Pakin and S.Coll , "STROM :Lightning –Fast Resource Management ",in proceedings of the IEEE/ACM SC2002 Conference, pp 16-22,Nov 2002.

[12] D.G.Feitelson and L.Rudolph ,. "Metric and Benchmarking for Parallel Job Scheduling ",Job Scheduling Strategies for parallel Processing in Springer Berlin ,Volume 1459/1998 pp 1-24, July 2006.

[13] S.Srinivasan ,R.Kettimathu,V.Subramani and P.Sadayappan , "Selective Reservation Strategies for Backfilling Job Scheduling ",Job Scheduling Strategies for Parallel Processing ,PP 55-71, 2002.

[14] D.S.Nikolopoulos and C.D.Polychronopoulos , "Adaptive Scheduling Under Memory Constraints on Non-Dedicated Computational Farme ,"Future Generation Computer Systems,Vol 19,no 4,pp 505-519,May 2003.

[15] D.G.Feitelson .Logs of real parallel workloads from production systems.http://www.c s.hiji.ac.il/labs/parallel/workload/logs.html. The access dates are Oct 1994- Dec 1993 for Fine grain Workloads. The access dates are January 2007 to June 2007 for the Medium grain Workloads .The access dates are June 2006 to September 2006.The access dates are August 2004 to May 2005.