Ain Shams University

كلية الحاسبات والمعلومات

Volume 4, Number 1, January 2004

INTERNATIONAL JOURNAL OF INTELLIGENT COMPUTING AND INFORMATION SCIENCES

EDITORS

MOHAMED F. TOLBA

MOHAMED S. ABDEL-WAHAB

ABDEL-BADEEH M. SALEM

Faculty of Computer & Information Sciences
Ain Shams University, Cairo, Egypt.

www.fcisainshams.net
e-mail: ijicis@fcisainshams.net

INTERNATIONAL JOURNAL OF INTELLIGENT COMPUTING AND INFORMATION SCIENCES

Volume 4, Number 1, January 2004

IJICIS, VOL. 4, NO. 1, January 2004

EVALUATION OF HFT TECHNIQUE PERFORMANCE USING TASK-SCHEDULING ALGORITHMS

O. A. Abulnaja

Department of Computer Science, King Abdulaziz University
Jeddah, Saudi Arabia
email: *abulnaja@kaau.edu.sa*

Abstract: *In earlier work we have introduced an efficient hardware fault-tolerant approach for reliable execution of tasks. The proposed approach called the Hardware Fault-Tolerant (HFT) approach. Also, we have introduced the concept of dynamic group maximum matching, which is used to group nodes of a graph into disjoint groups with different sizes dynamically. Furthermore, we have proposed the Dynamic Group Maximum Matching (DGMM) algorithm for finding the dynamic group maximum matching. In addition, we have proposed several hardware fault-tolerant scheduling algorithms, based on the HFT technique and the DGMM algorithm.*

In this work, we studied the effect of the HFT technique on system performance for four of the proposed scheduling algorithms. The studied algorithms are: Hardware Fault-Tolerant (FCFS + First Disagreement Graph First + First Started First), Hardware Fault-Tolerant (FCFS + First Fit First + First Disagreement Graph First + First Started First), Hardware Fault-Tolerant (FCFS + First Disagreement Graph First + Largest Group First), and Hardware Fault-Tolerant (FCFS + First Fit First + First Disagreement Graph First + Largest Group First) scheduling algorithms. Two performance metrics were evaluated. The first metric is system mean response time. The second metric is percentage of tasks of a certain type completed.

Keywords: *Performance Evaluation; Fault Tolerance; Fault Diagnosis; Task Scheduling; Networks.*

1. Introduction

In recent years, computer systems have been used in many applications ranging from critical applications to industrial process control and business applications. However, reliable execution of tasks and automatic detection and identification of faulty components where faulty outputs can jeopardize human lives or can cause loss of money are important design issues.

The first system-level fault diagnosis model in multicomputer systems was introduced in [1]. Another diagnostic system-level model was proposed in [2]. Both models assume that processors test each other by sending stimuli and waiting for a response. Due to several limitations related to this approach, other approaches were developed [3]. Among those approaches is the comparison-based model [4, 5], this approach assumes that each task is assigned to two different processors in the system for execution and a central observer compares their outputs. Then, the comparison-based approach is extended to allow the comparison of the outputs to take place in the processors themselves [6], but still the outcomes of the comparisons need to be sent to a central observer for diagnosis. In [7], the researchers allowed one of the processors being compared to be the comparator processor.

1.1. Dynamic Group Maximum Matching Concept

The maximum number of hardware faults that a system can tolerate with respect to a task T_i is defined as the task *hardware reliability degree* t_i. As a task hardware reliability degree increases, more redundancy is used. In [8, 9], the researchers assumed that all the tasks running in the system have equal hardware reliability degree t, and they partitioned the system into groups of size $(t + 1)$.

In [10], we have introduced the *dynamic group maximum matching* concept, where the system is partitioned into disjoint groups with different sizes dynamically. When a task T_i with the hardware reliability degree t_i is scheduled by the scheduler for execution, a group of processors of size $g_i = t_i + 1$ is assigned to the task. We also have proposed the *Dynamic Group Maximum Matching* (*DGMM*) algorithm for finding the dynamic group maximum matching.

The DGMM algorithm works as follows. When a task T_i with a group size g_i is scheduled for execution, the DGMM algorithm is called to find a connected subgraph G_i of size g_i in the system graph. The DGMM algorithm starts grouping processors by finding a free processor with the lowest degree in the system graph, adding it to the group G_i, and then finding a free neighboring processor of the group G_i with the lowest degree and adding it to the group G_i and so on. The DGMM algorithm returns either a group G_i with size equal to g_i, if possible, or a group G_i with a size smaller than g_i. The formal algorithm and an example are given in [10]

1.2. Hardware Fault-Tolerant (HFT) Technique

In [10], we have developed the Hardware Fault-Tolerant (HFT) technique. The technique is devised for the reliable execution of tasks and concurrent on-line fault diagnosis, where processors and communication channels are subject to failures. For reliable execution of tasks each task is assigned to a group of processors. Processors are grouped using the concept of dynamic group maximum matching. A task output is released if at least $((t_i + 1)$ - *number of diagnosed faults*) processors agree with each other on the output for the task T_i, where t_i is an upper bound on the number of faulty processors and communication channels the system can tolerate with respect to the task T_i (i.e., task T_i hardware reliability degree).

The HFT technique, in contrast to the most existing works that have focused mainly on improving the system reliability without any attempt to improve the system performance, attempts to maximize the system performance concurrently.

1.2.1. Comparison Model of Computation

When two neighboring processors P_i and P_j which are assigned to execute a task T_k finish executing the assigned task, first they exchange and compare their outputs and then each processor P_i (P_j) obtains its test outcome a_{ij} (a_{ji}) for the assigned task as follows:

1. If the processor P_i (P_j) agrees with the processor P_j (P_i) then
   (a) $a_{ij} = 0$ $(a_{ji} = 0)$

2. Else
(b) $a_{ij} = 1$ ($a_{ji} = 1$)

Remarks:
1. a_{ij} and a_{ji} may not be the same.
2. A faulty processor, a faulty channel or both could be the source of the disagreement between the two processors.
3. Processors P_i and P_j may produce the same output and agree with each other on the output, even if one (or both) of them are faulty which depends on whether faults in the processors and/or communication channel between them affected their outputs or not.

1.2.2. Disagreement Graph

A disagreement graph DG_i (N_i, E_i), where N_i is the set of nodes of the DG_i and E_i is the set of edges of the DG_i, with respect to a task T_i is obtained as follows. Every node $X \in N_i$ contains some processors of the group G_i that agree with each other on the output for the task T_i. An edge exists between two nodes $X \in N_i$ and $Y \in N_i$ if there exists a disagreement between a processor in node X and a processor in node Y over the output of the task T_i, provided that those processors are neighbors in the system graph. Agreement operation has a *transitivity* property. That is if P_i and P_j agree with each other on the output for the task T_i and in turn P_j and P_k agree with each other on the output for the task T_i, then P_i and P_k agree with each other on the output for the task T_i.

An illustrating example of the HFT technique is given in [10].

1.3. Hardware Fault-Tolerant Scheduling Algorithms

In [11, 12], we have introduced several hardware fault-tolerant scheduling algorithms. These scheduling algorithms are based on the Hardware Fault-Tolerant (HFT) technique and the Dynamic Group Maximum Matching (DGMM) algorithm. In this paper, due to the space limitations, we will limit our study only to the following four scheduling algorithms.

1.3.1. Hardware Fault-Tolerant (FCFS + First Disagreement Graph First + First Started First) Scheduling Algorithm

The Hardware Fault-Tolerant (FCFS + First Disagreement Graph First + First Started First) (*FCFSFDGFFSF)* scheduling policy works as follows. As tasks arrive at the system, they are queued up along with their group sizes (i.e. g_i = task hardware reliability degree $t_i + 1$) in a single task queue Q. When a task T_i is scheduled for execution, the DGMM algorithm is called to find a group of the required size. If the returned subgraph is equal to the required group size, the task is assigned to the subgraph for execution; otherwise, the DGMM algorithm is called to find a new subgraph of the required size in a different part of the system graph.

This process is repeated until either a group of the required size is obtained or the entire system graph is searched without success. In the latter case, the task T_i is inserted in the aborted task queue Q_a for later execution. In the former case, the task T_i is assigned to all processors in the group G_i for

execution. When a task T_i completes its execution by all the processors in the group G_i, neighboring processors exchange and compare their outputs. Then the disagreement graph for the task is obtained.

A task is released if at least $((t_i + 1)$ - *number of diagnosed faults)* processors in the group agree with each other on the output of the task. Otherwise; the group size is incremented by one ($g_i = g_i + 1$) and the DGMM algorithm is called to add one more neighboring processor to the group, if possible. If not, the FCFSFDGFFSF algorithm finds a neighboring task T_j that has no disagreement graph yet. If such a task exists then the task T_j is aborted and added to the aborted task queue Q_a, for later execution, the group G_j is returned to the system graph and the DGMM algorithm is called. In case of ties, select the task T_j that has started its execution the latest.

In case of ties, select a task randomly. If no such task exists and the task T_i has started its execution before its neighbors, then the FCFSFDGFFSF algorithm finds a neighboring task T_j that has started its execution the latest. If such a task exists then the task T_j is aborted and added to the aborted task queue Q_a, for later execution, the group G_j is returned to the system graph and the DGMM algorithm is called.

In case of ties, select a task randomly. If no such task exists then, the DGMM algorithm is called to find another subgraph of size g_i in a different part of the system graph. Calling the DGMM algorithm is repeated until either a group of size g_i is obtained or the entire system graph is searched without success. In the latter case, the task T_i is aborted and inserted in the aborted task queue Q_a for later execution. In the former case, the task is assigned to the group G_i for execution. The above process is repeated until either the task is aborted and inserted in the aborted task queue Q_a for later execution or the output of the task is obtained. The formal algorithm is given in [11].

1.3.2. Hardware Fault-Tolerant (FCFS + First Fit First + First Disagreement Graph First + First Started First) Scheduling Algorithm

The Hardware Fault-Tolerant (FCFS + First Fit First + First Disagreement Graph First + First Started First) *(FCFSFFFFDGFFSF)* scheduling policy works as follows. As tasks arrive at the system, they are queued up along with their group sizes (i.e. g_i = task hardware reliability degree $t_i + 1$) in a single task queue Q. When a task T_i is scheduled for execution the DGMM algorithm is called to find a group of the required size. If the returned group size by the DGMM algorithm is smaller than the required group size, then the returned group is allocated to the first task T_j in the task queue Q that fits the returned group. Next, the DGMM algorithm is called to find another subgraph of size g_i in a different part of the system graph to allocate the task T_i.

This process is repeated until either a group of size g_i is obtained or the entire system graph is searched without success. In the latter case, the task T_i is added to the aborted task queue Q_a for later execution. In the former case, the task T_i is assigned to all processors in the returned group G_i for execution. When a task T_i completes its execution by all the processors of its group G_i, neighboring processors exchange and compare their outputs. Then the disagreement graph for the task is obtained.

A task T_i is released if at least $((t_i + 1)$ - *number of diagnosed faults)* processors in G_i agree with each other on the output for the task. Otherwise, the task group size is incremented by one ($g_i = g_i + 1$) and the DGMM algorithm is called to add one more neighboring processor to the group G_i. If the returned subgraph is equal to the required group size, the task T_i is assigned to all the processors in the

group G_i. Otherwise, the FCFSFFFFDGFFSF algorithm finds a neighboring task T_j that has no disagreement graph yet. If such a task exists, then abort the task T_j and add it to the tail of the aborted task queue Q_a for later execution, the group G_j of the task T_j is returned to the system graph and the DGMM algorithm is called.

In case of tie select a task T_j that has started its execution the latest. In case of ties, select a task randomly. If no such task exists and the task T_i has started its execution before its neighbors, then the FCFSFFFFDGFFSF algorithm finds a neighboring task T_j that has started its execution the latest. If such a task exists, then abort the task T_j and add it to the tail of the aborted task queue Q_a for later execution, the group G_j of the task T_j is returned to the system graph and the DGMM algorithm is called.

In case of ties, select a task randomly. If no such task exists, then the DGMM algorithm is called to find another subgraph of size g_i in a different part of the system graph to allocate the task T_i. Calling the DGMM algorithm is repeated until either a group of size g_i is obtained or the entire system graph is searched without success. In the latter case, the task T_i is aborted and added to the aborted task queue Q_a for later execution. In the former case, the task T_i assigned to all the processors the group G_i for execution. The above process repeated until either the task is aborted and added to the aborted task queue Q_a or the output of the task is obtained. The formal algorithm is given in [11].

1.3.3. Hardware Fault-Tolerant (FCFS + First Disagreement Graph First + Largest Group First) Scheduling Algorithm

The Hardware Fault-Tolerant (FCFS + First Disagreement Graph First + Largest Group First) *(FCFSFDGFLGF)* scheduling policy works as follows. As tasks arrive at the system, they are queued up along with their group sizes (i.e., g_i = task hardware reliability degree $t_i + 1$) in a single task queue Q. When a task T_i is scheduled for execution, the DGMM algorithm is called to find a group of the required size. If the returned subgraph is equal to the required group size, the task is assigned to the subgraph for execution; otherwise, the DGMM algorithm is called to find a new subgraph of the required size in a different part of the system graph.

This process is repeated until either a group of the required size is obtained or the entire system graph is searched without success. In the latter case, the task T_i is inserted in the aborted task queue Q_a for later execution. In the former case, the task T_i is assigned to all processors in the group G_i for execution. When a task completes its execution by all the processors in the group G_i, neighboring processors exchange and compare their outputs. Then the disagreement graph for the task is obtained.

A task T_i is released if at least $((t_i + 1)$ - *number of diagnosed faults)* processors in G_i agree with each other on the output for the task. Otherwise, the group size is incremented by one ($g_i = g_i + 1$) and the DGMM algorithm is called to add one more neighboring processor to the group, if possible. If not, the FCFSFDGFLGF algorithm finds a neighboring task T_j that has no disagreement graph yet. If such a task exists, then the task T_j is aborted and inserted at the tail of the aborted task queue Q_a for later execution, the group G_j is returned to the system graph, and the DGMM algorithm is called.

In case of ties, select the task that has the smallest group size. If no such task exists and the task T_i has the largest group size among its neighbors, then the FCFSFDGFLGF algorithm finds a neighboring task T_j that has the smallest group size. If such a task exists, then the task T_j is aborted and

inserted at the tail of the aborted task queue Q_a for later execution, the group G_j is returned to the system graph, and the DGMM algorithm is called.

In case of ties, select a task randomly. If there is no such task exists then the DGMM algorithm is called to find another subgraph of size g_i to allocate the task T_i in a different part of the system graph. Calling the DGMM algorithm is repeated until either a group of size g_i is obtained or the entire system graph is searched without success. In the latter case, the task T_i is aborted and added to the aborted task queue Q_a. In the former case, the task T_i is assigned to all the processors in the group G_i for execution. The above process is repeated until either the task is aborted and added to the aborted task queue Q_a or the task output is obtained. The formal algorithm is given in [12].

1.3.4. Hardware Fault-Tolerant (FCFS + First Fit First + First Disagreement Graph First + Largest Group First) Scheduling Algorithm

The Hardware Fault-Tolerant (FCFS + First Fit First + First Disagreement Graph First + Largest Group First) *(FCFSFFFFDGFLGF)* scheduling policy works as follows. As tasks arrive at the system , they are queued up along with their group sizes (i.e., g_i = task hardware reliability degree $t_i + 1$) in a single task queue Q. When a task T_i is scheduled for execution the DGMM algorithm is called to find the required group size for the task. If the returned group size by the DGMM algorithm is smaller than the required group size, then the returned group is allocated to the first task T_j in the task queue Q that fits the returned group. Next, the DGMM algorithm is called to find another subgraph of size g_i in a different part of the system graph to allocate the task T_i.

This process is repeated until either a group of size g_i is obtained or the entire system graph is searched without success. In the latter case, the task T_i is added to the aborted task queue Q_a for later execution. In the former case, the task T_i is assigned to all processors in the returned group G_i for execution. When a task T_i completes its execution by all the processors of its group G_i, neighboring processors exchange and compare their outputs. Then the disagreement graph for the task is obtained.

A task T_i is released if at least $((t_i + 1)$ - *number of diagnosed faults)* processors in G_i agree with each other on the output for the task. Otherwise, the task group size is incremented by one ($g_i = g_i + 1$) and the DGMM algorithm is called to add one more neighboring processor to the group G_i, if possible. If not, the FCFSFFFFDGFLGF algorithm finds a neighboring task T_j that has no disagreement graph yet. If such a task exists, then the task T_j is aborted and added to the aborted task queue Q_a for later execution, the group G_j is returned to the system graph and the DGMM algorithm is called. In case of ties, select the task T_j that has the smallest group size.

In case of ties, select a task randomly. If no such task exists and the task T_i has the largest group size among its neighbors, then the FCFSFFFFDGFLGF algorithm finds a neighboring task T_j that has the smallest group size. If such a task exists, then the task T_j is aborted and added to the aborted task queue Q_a for later execution, the group G_j is returned to the system graph and the DGMM algorithm is called.

In case of ties, select a task randomly. If no such task exists, then the DGMM algorithm is called to find another subgraph of size g_i to allocate the task T_i in a different part of the system graph. Calling the DGMM algorithm is repeated until either a group of size g_i is obtained or the entire system graph is searched without success. In the latter case, the task T_i is aborted and inserted in the aborted task queue

Q_a for later execution. In the former case, the task T_i is assigned to all processors in the group G_i for execution. The above process is repeated until either the task is aborted and added to the aborted task queue Q_a or the task output is obtained. The formal algorithm is given in [12].

2. Simulation Model

The features of the simulator are summarized as follows [13]:

1. The computing environment is an $M \times M$ *torus* system ($M \geq 1$) connected to a host machine where scheduling and obtaining tasks disagreement graphs take place.
2. Each task (program) T_i arrives at the system (along with its reliability degree t_i) will be assigned to a group G_i of size g_i (initially $g_i = t_i + 1$).
3. Tasks interarrival times are exponentially distributed with the average arrival rate λ.
4. Tasks mean execution times are exponentially distributed. Tasks arrived at the system could be of any type; in other words, tasks could have different mean execution times.

3. Simulation Results

In our simulation we consider a 6×6 torus system ($M = 6$), see Figure 1. We assume that there are long tasks and short tasks. Mean execution time of long tasks is 10 *time units* and mean execution time of short tasks is 1 *time unit*. Tasks arrive at the system with the probability of being long task equal to (X) and of short task equal to (1-X); in other words, task length probability has a *Bernoulli* probability distribution. All the results given in this work assume $X = 0.5$. Also, we assume that there are five types of tasks hardware reliability degrees: $type_0$ ($t_i = 0$), $type_1$ ($t_i = 1$), $type_2$ ($t_i = 2$), $type_3$ ($t_i = 3$) and $type_4$ ($t_i = 4$).

Tasks arrive at the system with the probability of being of $type_i$ equal to (0.2); in other words, tasks hardware reliability degrees probability has a *Binomial* probability distribution. Each processor in the system has the probability (reliability) of being fault-free equal to (R_p); in other words, a processor reliability has a Bernoulli probability distribution. Each communication link in the system has the probability of being fault-free equal to (R_l); in other words, a communication link reliability has also a Bernoulli probability distribution. In our simulation, we consider four failure cases. First case, processors and communication links are fault-free, $R_p = 1$, $R_l = 1$. Second case, only communication links are subject to failures, $R_p = 1$, $R_l = 0.9$. Third case, only processors are subject to failures, $R_p = 0.9$, $R_l = 1$. Fourth case, both processors and communication links are subject to failures, $R_p = 0.9$, $R_l = 0.9$.

Our simulation terminates when the number of tasks released by the system is equal to 3000 tasks. The first 300 tasks released by the system are discarded, so the initial transient state of the system does not affect the simulation results. Each performance metric reading is an average over 10 runs. We evaluate two performance metrics. The first metric is system mean response time. The second metric is number of tasks of type completed, for i = 0, 1, 2, 3, 4.

3.1. FCFSFDGFFSF Scheduling Algorithm Performance

Figure 2 shows system average response time under the Hardware Fault-Tolerant (FCFS + First Disagreement Graph First + First Started First) (FCFSFDGFFSF) scheduling algorithm. In Figure 2, up to a point as task arrival rate λ increases the system average response time also increases (in our experiment $\lambda = 3$ for the last failure case, and $\lambda = 4$ for the first three failure cases). Beyond that point, as arrival rate λ increases, the system average response time decreases. This is due to the fact that when the task arrival rate λ is high, more tasks will be queued up in the task queue, thus the probability that DGMM algorithm will find the required group size for tasks with large group size is much lower than the probability of finding the required group size for tasks with small group sizes. Thus, FCFSFDGFFSF scheduling algorithm will abort tasks with large group size in the aborted task queue. This means that tasks with small group sizes will be executed first, i.e., more tasks will be executed on the system.

With a higher task arrival rate λ, we expect the system average response time will increase. This is due to that the fact the length of the task queue will grow longer. Thus, even tasks with small group size have to wait longer in the system queue before being scheduled for execution. Also from the plot we can see that as arrival rate λ increases, the difference in the performance between the four failure cases decreases. This is due to the fact that when the task arrival rate λ is high, more tasks will be queued up in the task queue, thus the probability that DGMM algorithm will find the required group size for tasks with large group size is much lower than the probability of finding the required group size for tasks with small group sizes. Thus, FCFSFDGFFSF scheduling algorithm will abort tasks with large group size in the aborted task queue. This means that both faulty systems and fault-free systems will execute tasks with small group sizes. In other words, with a higher task arrival rate λ faulty components have little impact on the system performance.

Figures 3, 4, 5 and 6 show the number of tasks of $type_i$ completed, for $i = 0, 1, 2, 3, 4$, by FCFSFDGFFSF scheduling algorithm, under the four failure cases respectively. From these plots we can see that when task arrival rate λ is equal 2, the number of tasks completed for all tasks types under each one of the failure cases is almost the same. This due the fact that the system is not crowded, thus, the probability that DGMM algorithm will find the required group size for any task is high.

Also, we can see in as the task arrival rate λ increases (in our experiment arrival rate $\lambda > 2$), the number of tasks of $type_i$ completed, for $i = 0, 1$, increases and the number of tasks of $type_i$ completed, for $i = 2, 3, 4$, decrease. This is due to the fact that when the task arrival rate λ is high, more tasks will be queued up in the task queue, thus the probability that DGMM algorithm will find the required group size for tasks with large group size is much lower than the probability of finding the required group size for tasks with small group sizes. Thus, FCFSFDGFFSF scheduling algorithm will abort tasks with large group size in the aborted task queue. This means that tasks with small group sizes will be executed first, in other words, FCFSFDGFFSF scheduling algorithm favors tasks with small group sizes over tasks with large group sizes for execution.

Also, from Figures 3, 4, 5 and 6 we can see that the number of tasks of $type_i$ completed, for $i = 0, 1$ increases and the number of tasks of $type_i$ completed, for $i = 2, 3, 4$ decreases when the number of faulty components increases.

3.2. FCFSFFFFDGFFSF Scheduling Algorithm Performance

Figure 7 shows system average response time under the Hardware Fault-Tolerant (FCFS + First Fit First + First Disagreement Graph First + First Started First) (FCFSFFFFDGFFSF) scheduling algorithm. In Figure 7, up to a point as task arrival rate λ increases the system average response time also increases (in our experiment $\lambda = 3$ for the last two failure cases, and $\lambda = 4$ for the first two failure cases). Beyond that point, as arrival rate λ increases, the system average response time decreases. This is due to the fact that when the task arrival rate λ is high, more tasks will be queued up in the task queue, thus the probability that DGMM algorithm will find the required group size for tasks with large group size is much lower than the probability of finding the required group size for tasks with small group sizes. Thus, FCFSFFFFDGFFSF scheduling algorithm will abort tasks with large group size in the aborted task queue. This means that tasks with small group sizes will be executed first, i.e., more tasks will be executed on the system.

With a higher task arrival rate λ (in our experiment arrival rate $\lambda \geq 5$ for the last two failure cases, and $\lambda \geq 6$ for the first two failure cases), the system average response time will increase. This is due to that the fact the length of the task queue will grow longer. Thus, even tasks with small group size have to wait longer in the system queue before being scheduled for execution.

Figures 8, 9, 10 and 11 show the number of tasks of $type_i$ completed, for $i = 0, 1, 2, 3, 4$, by FCFSFFFFDGFFSF scheduling algorithm, under the four failure cases respectively. We can see that when task arrival rate λ is equal 2, the number of tasks completed for all tasks types under each one of the failure cases is almost the same. This is due to the fact that the system is not crowded, thus, the probability that DGMM algorithm will find the required group size for any task is high. Beyond that point, in Figures 8 and 9 we can see as the task arrival rate λ increases (in our experiment arrival rate $\lambda \geq 6$) the number of tasks of $type_i$ completed, for $i = 1, 2$, decreases and the number of tasks of $type_i$ completed, for $i = 0, 3, 4$, increases.

From Figures 10 and 11 we can see as the task arrival rate λ increases (in our experiment arrival rate $\lambda \geq 6$) the number of tasks of $type_i$ completed, for $i = 0, 1, 2$, decreases and the number of tasks of $type_i$ completed, for $i = 3, 4$, increases. This means that the system average response time will be high, see Figure 7.

Also, from Figures 8, 9, 10 and 11 we can see that the number of tasks of $type_i$ completed, for $i = 3, 4$ increases and the number of tasks of $type_i$ completed, for $i = 0, 1, 2$ decreases when the number of faulty components increases.

3.3. FCFSFDGFLGF Scheduling Algorithm Performance

Figure 12 shows system average response time under the Hardware Fault-Tolerant (FCFS + First Disagreement Graph First + Largest Group First) (FCFSFDGFLGF) scheduling algorithm. In Figure 12, up to a point as task arrival rate λ increases the system average response time also increases (in our experiment $\lambda = 3$ for the last failure case, and $\lambda = 4$ for the first three failure cases). Beyond that point, as arrival rate λ increases, the system average response time decreases. This is due to the fact that when the task arrival rate λ is high, more tasks will be queued up in the task queue, thus the probability that DGMM algorithm will find the required group size for tasks with large group size is much lower

than the probability of finding the required group size for tasks with small group sizes. Thus, FCFSFDGFLGF scheduling algorithm will abort tasks with large group size in the aborted task queue. This means that tasks with small group sizes will be executed first, i.e., more tasks will be executed on the system.

Also, from the plot we can see that as the task arrival rate λ increases, the difference in the performance between the four failure cases decreases. This is due to the fact that when the task arrival rate λ is high, more tasks will be queued up in the task queue, thus the probability that DGMM algorithm will find the required group size for tasks with large group size is much lower than the probability of finding the required group size for tasks with small group sizes. Thus, FCFSFDGFLGF scheduling algorithm will abort tasks with large group size in aborted task queue. This means that tasks with small group sizes will be executed first, i.e., more tasks will be executed on the system. Thus, both faulty systems and fault-free systems will execute tasks with small group sizes. In other words, with a higher task arrival rate λ faulty components have little impact on the system performance.

Figures 13, 14, 15 and 16 show the number of tasks of $type_i$ completed, for $i = 0, 1, 2, 3, 4$, by FCFSFDGFLGF scheduling algorithm, under the four failure cases respectively. From theses plots we can see that when task arrival rate λ is equal 2, the number of tasks completed for all tasks types under each one of the failure cases is almost the same. This due the fact that the system is not crowded, thus, the probability that DGMM algorithm will find the required group size for any task is high. Also, we can see in as the task arrival rate λ increases (in our experiment arrival rate $\lambda > 2$), the number of tasks of $type_i$ completed, for $i = 0, 1$, increases and the number of tasks of $type_i$ completed, for $i = 2, 3, 4$, decrease.

Furthermore, we can see that as the number of faulty components increases the number of tasks of $type_i$ completed, for $i = 0, 1$, also increases and the number of tasks of $type_i$ completed, for $i = 2, 3, 4$, decrease. This is due to the fact that when the system has some faulty components, the probability that DGMM algorithm will find the required group size for tasks with large group size is much lower than the probability of finding the required group size for tasks with small group sizes. Thus, FCFSFDGFLGF scheduling algorithm will abort tasks with large group sizes in the aborted task queue. This means that tasks with small group sizes will be executed first.

3.4. FCFSFFFFDGFLGF Scheduling Algorithm Performance

Figure 17 shows system average response time under the Hardware Fault-Tolerant (FCFS + First Fit First + First Disagreement Graph First + Largest Group First) (FCFSFFFFDGFLGF) scheduling algorithm. In Figure 17, up to a point as task arrival rate λ increases the system average response time also increases (in our experiment $\lambda = 3$ for the last two failure cases, and $\lambda = 4$ for the first two failure cases). Beyond that point, as arrival rate λ increases, the system average response time decreases. This is due to the fact that when the task arrival rate λ is high, more tasks will be queued up in the task queue, thus the probability that DGMM algorithm will find the required group size for tasks with large group size is lower than the probability of finding the required group size for tasks with small group sizes. Thus, FCFSFFFFDGFLGF scheduling algorithm will abort tasks with large group size in the aborted task queue.

With a higher task arrival rate λ (in our experiment arrival rate $\lambda \geq 5$ for the last three failure cases, and $\lambda \geq 6$ for the first failure case), the system average response time will increase. This is due to that the fact the length of the task queue will grow longer. Thus, even tasks with small group size have to wait longer in the system queue before being scheduled for execution.

Figures 18, 19, 20 and 21 show the number of tasks of $type_i$ completed, for $i = 0, 1, 2, 3, 4$, by FCFSFFFFDGFLGF scheduling algorithm under the four failure cases respectively. From these plots we can see that when task arrival rate λ is equal 2, the number of tasks completed for all tasks types under each one of the failure cases is almost the same. This due the fact that the system is not crowded, thus, the probability that DGMM algorithm will find the required group size for any task is high.

Beyond that point, in Figures 18 and 19, we can see that as the task arrival rate λ increases (in our experiment arrival rate $\lambda \geq 6$) the number of tasks of $type_i$ completed, for $i = 1, 2$, decreases and the number of tasks completed of $type_i$ completed, for $i = 0, 3, 4$, increases. From Figure 20 we can see as the task arrival rate λ increases (in our experiment arrival rate $\lambda \geq 6$) the number of tasks completed of $type_i$ completed, for $i = 0, 1$, decreases and the number of tasks of $type_i$ completed, for $i = 2, 3, 4$, increases. From Figure 21 we can see as the task arrival rate λ increases (in our experiment arrival rate $\lambda \geq 6$) the number of tasks completed of $type_i$ completed, for $i = 0, 1, 2$, decreases and the number of tasks of $type_i$ completed, for $i = 3, 4$, increases.

Also, from Figures 18, 19, 20 and 21 we can see that the number of tasks of $type_i$ completed, for $i = 3, 4$ increases and the number of tasks of $type_i$ completed, for $i = 0, 1, 2$ decreases when the number of faulty components increases.

4. Conclusion

In this work, via four scheduling algorithms, the performance of the Hardware Fault-Tolerant (HFT) was studied. Two performance metrics were evaluated: system mean response time and number of tasks of $type_i$ completed.

Under the Hardware Fault-Tolerant (FCFS + First Disagreement Graph First + First Started First) (FCFSFDGFFSF) scheduling algorithm and the Hardware Fault-Tolerant (FCFS + First Disagreement Graph First + Largest Group First) (FCFSFDGFLGF) scheduling algorithm, our simulation study showed that under the conditions experimented here, beyond a point, as arrival rate λ increases, the system average response time decreases. With a higher task arrival rate λ, we expect the system average response time will increase.

Under the Hardware Fault-Tolerant (FCFS + First Fit First + First Disagreement Graph First + First Started First) (FCFSFFFFDGFFSF) scheduling algorithm and the Hardware Fault-Tolerant (FCFS + First Fit First + First Disagreement Graph First + Largest Group First) (FCFSFFFFDGFLGF) scheduling algorithm, our simulation study showed that under the conditions experimented here, beyond a point, as arrival rate λ increases, the system average response time decreases. With a higher task arrival rate λ, as the arrival rate λ increases the system average response time also increases.

Also, the study showed that FCFSFDGFFSF and FCFSFDGFLGF scheduling algorithms perform almost the same. Furthermore, the study showed that the two scheduling algorithms average

response time outperforms the other algorithms average response time. On the other hand, FCFSFFFFDGFLGF scheduling algorithm gives the highest average response time.

In addition, the study showed that all the scheduling algorithms studied in this work favor tasks with small group sizes over tasks with large group sizes for execution.

References

[1] Preparata F. P., Metze G., Chien R. T., "On the Connection Assignment Problem of Diagnosable Systems," *IEEE Transactions on Electronic Computers*, vol. 16, December 1967, pp. 848-854.

[2] Barsi F., Grandoni F., Maestrini P., "A Theory of Diagnosability of Digital Systems," *IEEE Transactions on Computers*, vol. 25, June 1976, pp. 585-593.

[3] Dahbura A. T., "System-Level Diagnosis: A Perspective for The Third Decade," *Concurrent Computation: Algorithms, Architectures, Technologies*, Plenum, 1988.

[4] Malek M., "A Comparison Connection Assignment for Diagnosis of Multiprocessor Systems," in *Proceedings of 7th Int'l Symp. Computer Architecture*, 1980, pp. 31-36.

[5] Chwa K. Y., Hakimi S. L., "Schemes for Fault-Tolerant Computing: A Comparison of Modularly Redundant and *t*-Diagnosable Systems," *Information and Control*, vol. 49, 1981, pp. 212-238.

[6] Maeng J., Malek M., "A Comparison Connection Assignment for Self -Diagnosis of Multiprocessor Systems," *Digest 11th Int'l Symp. Fault Tolerant Computing*, 1981, pp. 173-175.

[7] Sengupta A., Dahbura A. T., "On Self-Diagnosable Multiprocessor Systems: Diagnosis by the Comparison Approach, " *IEEE Transactions on Computers*, vol. 41, November 1992, pp. 1386-1396.

[8] Hosseini S. H., "Fault-Tolerant Scheduling of Independent Tasks and Concurrent Fault-Diagnosis in Multiple Processor Systems," in *Proceedings of IEEE Int'l Conf. Parallel Processing*, Vol. I, Illinois, pp. 343-350, 1988.

[9] Hosseini S. H., Patel T. P., "An Efficient and Simple Algorithm for Group Maximum Matching," in *Proceedings of 4th ISMM/IASTED Int'l Conf. on Parallel and Distributed Computing Systems*, pp. 250-254, 1991.

[10] Abulnaja O. A., Hosseini S. H., Vairavan K., "Hardware Fault-Tolerant Scheduling Algorithms," in *Proceedings of 15th National Computer Conference*, Dhahran, Saudi Arabia, pp. 647 - 664, 1997.

[11] Abulnaja O. A., Hosseini S. H., Vairavan K., "Scheduling Algorithms for Reliable Execution of Tasks Under Hardware Faults," in *Proceedings of 7th Int'l Conf. Computer Theory & Applications*, Alexandria, Egypt, pp. 11.1 - 11.6, 1997.

[12] Abulnaja O. A., Hosseini S. H., Vairavan K., "On-Line System-Level Fault Diagnosis Scheduling Algorithms," in *Proceedings of 9th Int'l Conf. Computer Theory &*

Applications, Alexandria, Egypt, pp. 225 – 231, 1999.

[13] O. A. Abulnaja, "Evaluating the Effect of HFT Scheme on System Performance Via Priority Scheduling Algorithms Simulation," *Minufiya Journal of Electronic Engineering Research*, Faculty of Electronic Engineering, Minufiya University, vol. 13, no. 1, January 2003, pp. 91 – 104.

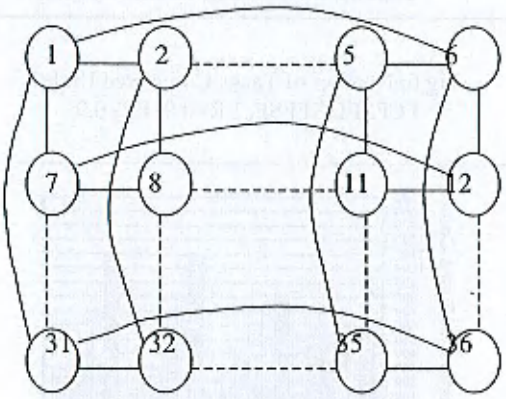

Fig 1: 6 × 6 Torus System

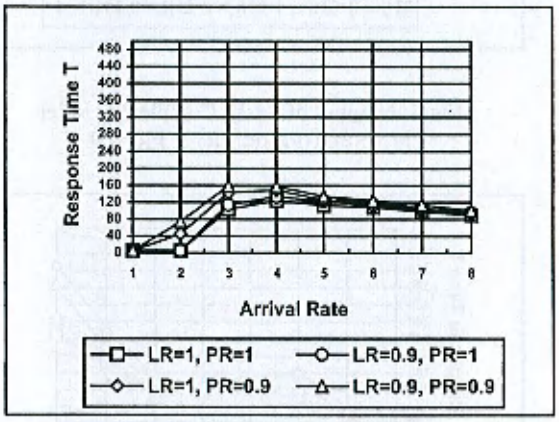

Fig 2: System Mean Response Time Under FCFSFDGFFSF

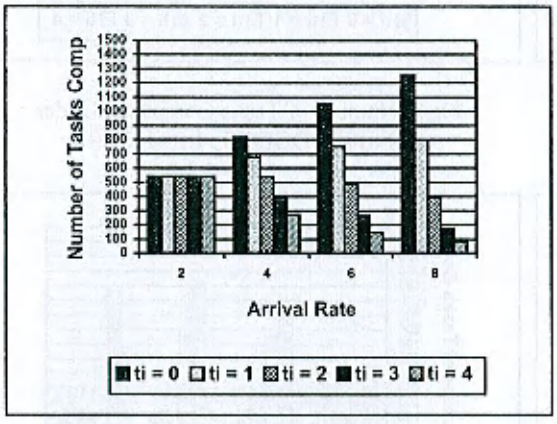

Fig 3: Number of Tasks Completed Under FCFSFDGFFSF, LR=1, PR=1

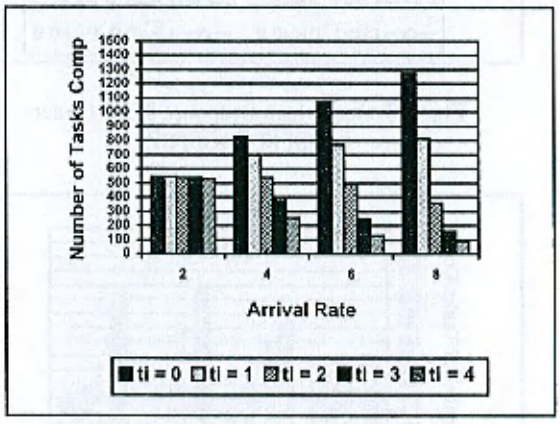

Fig 4: Number of Tasks Completed Under FCFSFDGFFSF, LR=0.9, PR=1

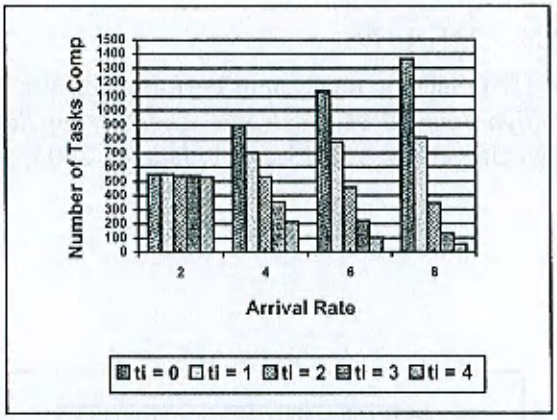

Fig 5: Number of Tasks Completed Under FCFSFDGFFSF, LR=1, PR=0.9

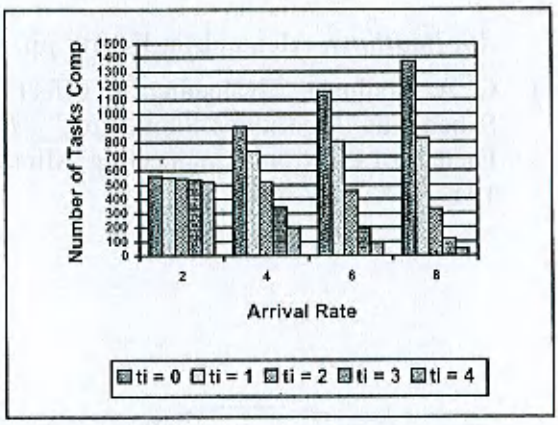

Fig 6: Number of Tasks Completed Under FCFSFDGFFSF, LR=0.9, PR=0.9

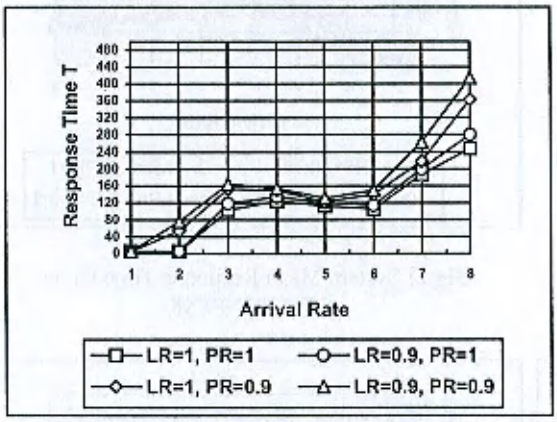

Fig 7: System Mean Response Time Under FCFSFFFFDGFFSF

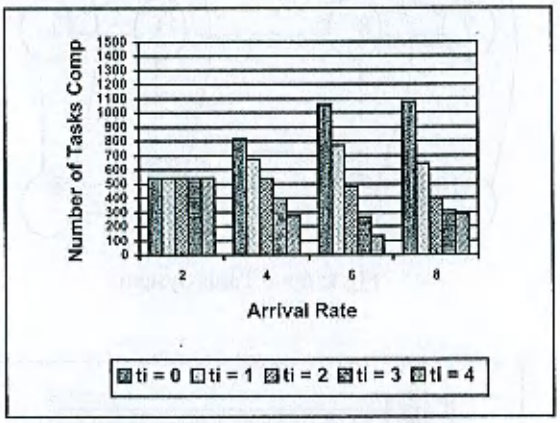

Fig 8: Number of Tasks Completed Under FCFSFFFFDGFFSF, LR=1, PR=1

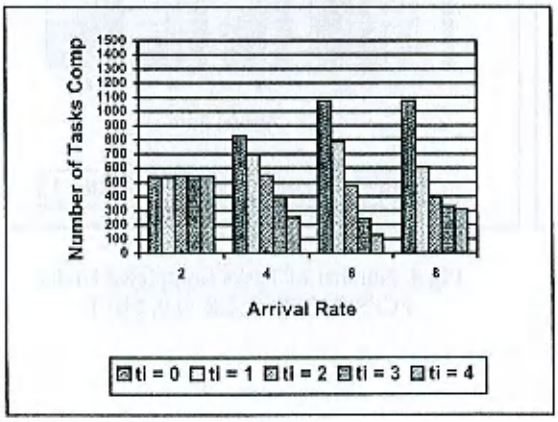

Fig 9: Number of Tasks Completed Under FCFSFFFFDGFFSF, LR=0.9, PR=1

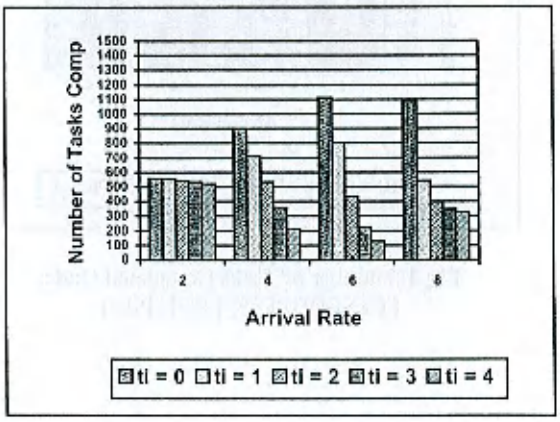

Fig 10: Number of Tasks Completed Under FCFSFFFFDGFFSF, LR=1, PR=0.9

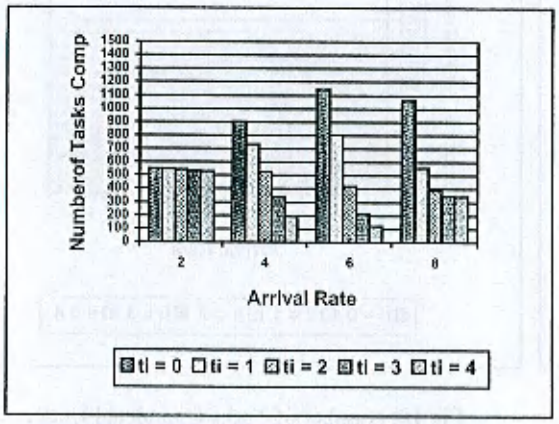

Fig 11: Number of Tasks Completed Under FCFSFFFFDGFFSF, LR=0.9, PR=0.9

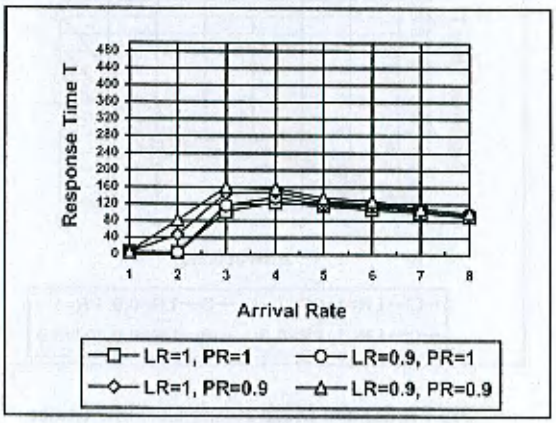

Fig 12: System Mean Response Time Under FCFSFDGFLGF

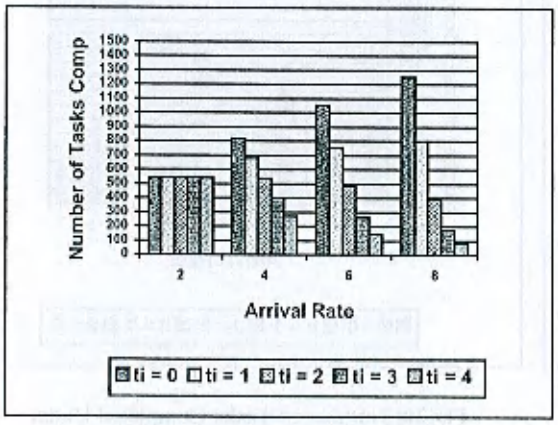

Fig 13: Number of Tasks Completed Under FCFSFDGFLGF, LR=1, PR=1

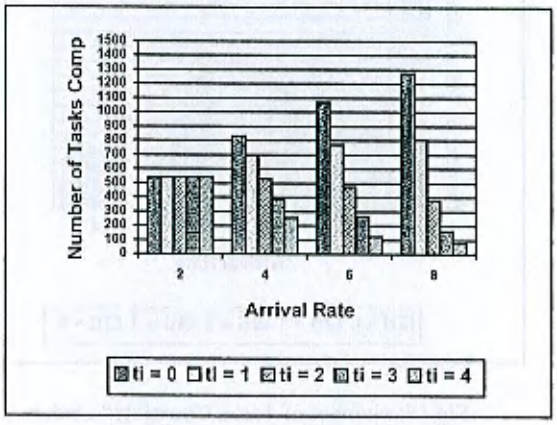

Fig 14: Number of Tasks Completed Under FCFSFDGFLGF, LR=0.9, PR=1

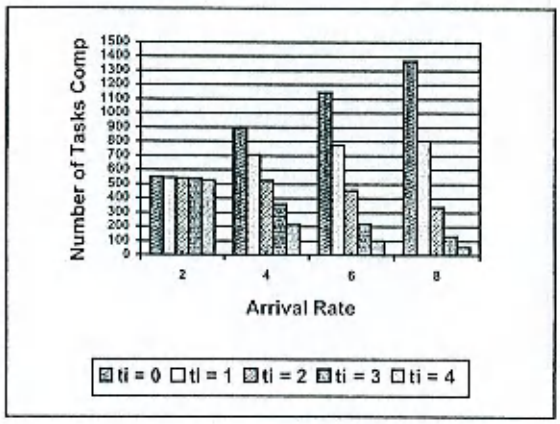

Fig 15: Number of Tasks Completed Under FCFSFDGFLGF, LR=1, PR=0.9

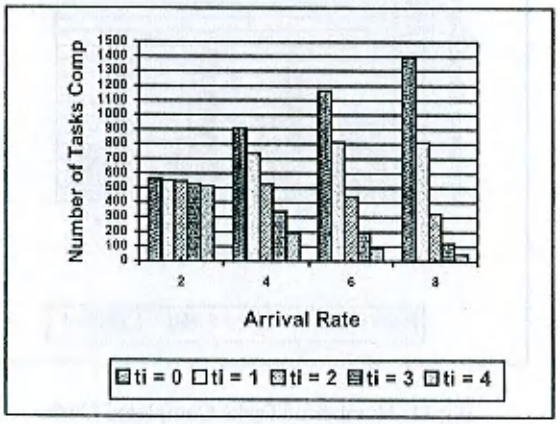

Fig 16: Number of Tasks Completed Under FCFSFDGFLGF, LR=0.9, PR=0.9

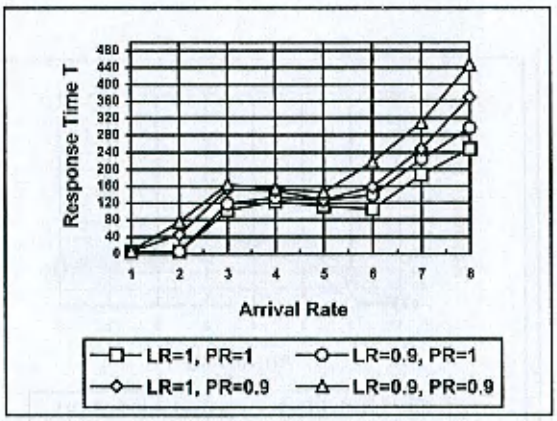

Fig 17: System Mean Response Time Under FCFSFFFFDGFLGF

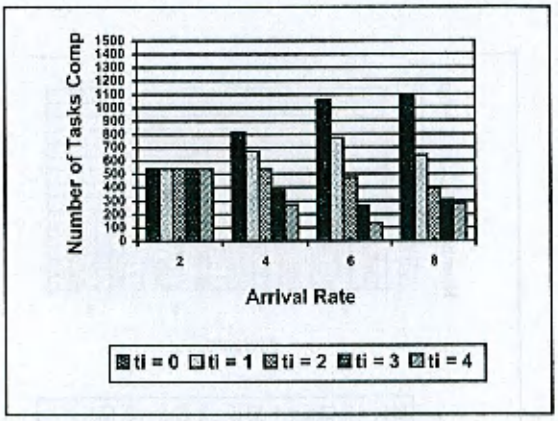

Fig 18: Number of Tasks Completed Under FCFSFFFFDGFLGF, LR=1, PR=1

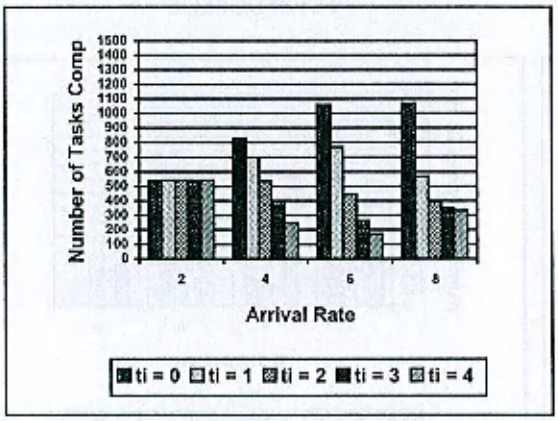

Fig 19: Number of Tasks Completed Under FCFSFFFFDGFLGF, LR=0.9, PR=1

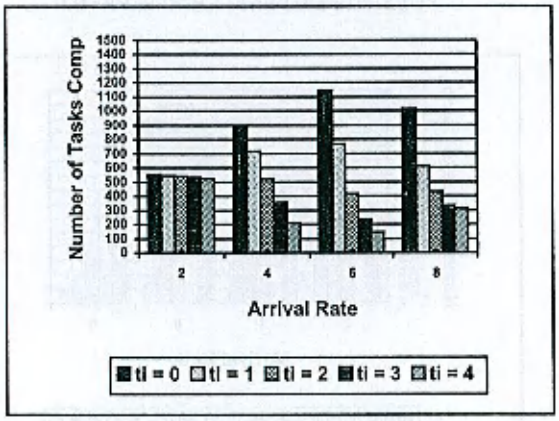

Fig 20: Number of Tasks Completed Under FCFSFFFFDGFLGF, LR=1, PR=0.9

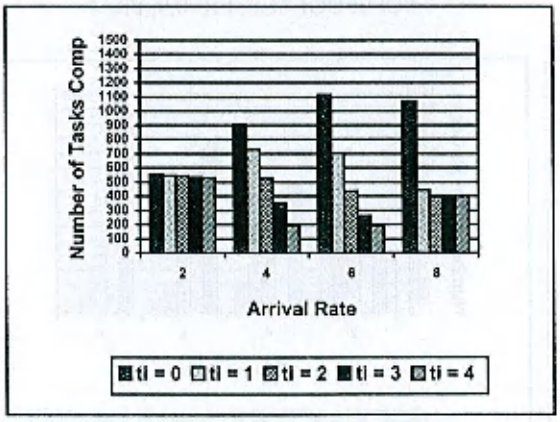

Fig 21: Number of Tasks Completed Under FCFSFFFFDGFLGF, LR=0.9, PR=0.9