special ssesion: Advanced information and wireless communication systems
session orginzier: Hesham Eldeeb

Scheduling Real Time Parallel Structure on Cluster Computing with Possible Processor failures

Alaa Amin and Reda Ammar
Computer Science and Eng. Dept.
University of Connecticut

Ayman El Dessouly
Electronics Research Institute
Dokki – Giza, Egypt

***Abstract*: -** Efficient task scheduling is essential for achieving high performance computing applications for distributed systems. Most of existing real-time systems consider schedulability as a main goal and ignores other effects such as machines failures. In this paper we develop an algorithm to efficiently schedule parallel task graphs (fork-join structures). Our scheduling algorithm considers more than one factor at the same time. These factors are scheduability, reliability of the participating processors and achieved degree of parallelism. To achieve most of these goals, we composed an objective function that combines these different factors simultaneously. The proposed objective function is adjustable to provide the user with a way to prefer one factor to the others. The simulation results indicate that our algorithm produces schedules where the applications deadlines are met, reliability is maximized and the application parallelism is exploited.

Key-Words: - Scheduling, Cluster Computing, Reliability, Real time, Parallel Structure, Heuristic search.

1 Introduction

Scheduling real time applications in a cluster environment is a challenging problem. Real time applications are composed of one or more tasks that are required to perform their functions under strict timing constraints (deadlines). These applications have to meet their deadlines amidst contradicting goals, while maximizing resource utilization. A task missing its deadline may result in a domino effect, possibly causing other tasks to miss their deadlines and resulting in a system failure. This system failure changes with the kind of real-time application. For instance, in hard real-time systems the effect of violating deadline may be catastrophic which indicate the necessity to be met. On the other side, in soft real-time systems, the utility of results produced by a task with soft deadline decreases over time after deadline expires which indicate that the system can withhold deadline violation [1].

There are several advantages of scheduling an application represented as task graphs instead of treating it as a single unit [12]. If the application is scheduled as one unit, it uses the same processor for a longer time and hence the chance of failure (hardware or software) increases. If the failure happens, the application may start over again and hence will not be able to finish by its deadline. On the other hand, each task has a shorter time and hence the probability of failure is smaller. In other words, if the application is scheduled as a single unit, its reliability is controlled by the reliability of one processor. Distributing parallel tasks among different processors may achieve better reliability and increase the degree of parallelism. However, there is an advantage of treating an application as one unit; the communication among its tasks is eliminated. This is true if the scheduler is able to find a processor with an enough processing power to satisfy the application's deadline. On the contrary, scheduling individual tasks on different processors will increase the possibility of having the required processing power due to their short life times although this will increase the remote communication time and may make the application subject to network failure. Hence scheduling applications' task graphs need to include several conflicting factors to find the best trade-off among these different them and to prove that it is better than treating an application as a single unit.

In this paper, we developed a scheduling method for parallel tasks of real-time applications. The algorithm achieves better performance than scheduling each application as a single unit. The paper is organized as follows. In section 2, related work is described. The scheduling problem and definitions are given in section 3. The new scheduling algorithm is presented in section 4 then followed by example in section 5. The simulation results are shown in section 6. Finally, section 7 provides the conclusion.

2 Related work

Performance and reliability are considered as significant requirements for real-time systems. In general, real-time applications are assumed either as a task graph [2-4]or as a single unit [5-7]. Most of the existing algorithms deal with the later kind of applications. High reliable real-time scheduler can be achieved by different ways. From the performance point of view; reliability could be taken into consideration as a factor when designing the scheduling algorithm from the very beginning[3, 4]. Another way is to use some techniques such as primary-backup to tolerate faults and consequently increase the system reliability [5, 6].

In [5], Manimaran Et.al proposed an algorithm for dynamically scheduling arriving independent real-time tasks with resources and primary-backup based fault tolerant requirements in a multiprocessor system. They didn't utilize processors available processing power or consider the reliability as a design factor.

In [6], Gosh et al. developed a fault-tolerance scheduling method for real-time systems that tolerate failure and consequently increase the system reliability. They took into account the processing power utilization of the resources. These methods can be applied on single unit application and it can't deal with a task graph application. Also, this method is designed for hard real-time applications.

In [7], Tsuchiya proposed a fault tolerant task scheduling techniques for real-time multiprocessor systems where aperiodic independent tasks arrive dynamically. Also, they didn't take the processing power utilization factor into consideration.

In [4], Qin et al. proposed a scheduling scheme with which real-time tasks with precedence constraints can statically be scheduled to tolerate the failure of one processor in a heterogeneous parallel and distributed system. They assume a heterogeneous system with reliable communication. The task model used is the one with precedence constraints. They used the greedy scheme EST to create the primary schedule. They didn't take processing power utilization into account and the algorithm is designed for hard real-time applications only.

In [3], Dogan and Özgüner proposed a reliable matching and scheduling algorithm. They introduced a cost function that combines schedulability and reliability at the same time. During the scheduling process, the cost function is checked and the maximum value is selected. The proposed method didn't take real-time constraints into consideration. Another heuristic method to determine an allocation that attempts to maximize the reliability is presented in [2]. The method is based on the concept of clustering (grouping) tasks to allocate tasks for maximizing reliability. However, they didn't consider real-time constraints.

All the above previous work didn't try to exploit the degree of parallelism included in the task graph in addition to satisfying real time constraints and maximizing reliability. Our algorithm combines these three factors together.

3 The Scheduling Problem and Definitions:

The system used for scheduling fork-join structures is a cluster computing. The cluster consists of a set P = $\{p_1, p_2, ..., p_n\}$ of identical processors. The processors of the cluster are fully connected by a real-time communication network that offers real-time communication guarantee [8]. This kind of network guarantees a reliable message passing system. A set of real-time applications (jobs) $\{A_1, A_2, A_3, .., A_m\}$. Each application can be modeled as control flow graph (CFG) G =(T, E), where T = $\{t_1, t2, t_3.,\ldots t_k)$ is a set of dependent tasks, and a set of edges E represents dependency relation among tasks. Each task v is characterized in terms of three attributes $\mathbf{\{t_i, d_i, s_i\}}$, where $\mathbf{t_i}$ is the task computation cost, $\mathbf{d_i}$ is the task deadline and $\mathbf{s_i}$ is the submission (start) time of the task. The starting time parameter determines the precedence relation among tasks. The tasks in this paper are parallel, dependent and represented in the form of a fork-join. Fig.1 shows an example of an application task graph that contains five parallel tasks along with their attributes.

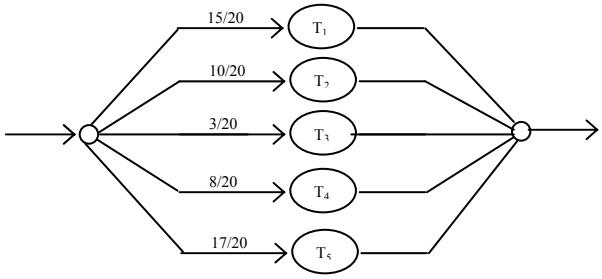


Fig 1 For-Join Example

Reliability computation: If a processor P_i executes task (t_j,d_j,s_j), the reliability of the is the processor is given by:

$$R = e^{-\lambda_i t_j} \qquad (1)$$

Where λ_i = processor **i** failure (hazard) rate.

t_j = task execution time

The failure rate represents how many failures per unit time could occur for the processor. The value of the failure rate depends on the processor. In this paper, we assume that the failure (hazard) rate is constant. This assumption has been widely used in computer systems performance and reliability analysis [3, 4, 9].

Using equation 1 we can calculate a single processor reliability when execute certain task. The goal now is to calculate the reliability of different structures and fork join structure in this paper. We use Fault tree

analysis [10] to achieve this goal. Fault-tree analysis is a deductive methodology for determining the potential causes of failures and for estimating the failure probabilities (consequently the reliability). Fault-tree analysis determines the causes of an undesired event, referred as the top event, since fault trees are drawn with it at the top of the tree. We can apply the fault tree approach to calculate the overall failure rate of a group of processors when execute a fork-join application. The following procedure describes the steps to calculate λ_p for the fork join structure application in fig.1.

Construct the fault tree for the application: Assume the application is assigned to the processors P_1, P_2, P_3, P_4, P_5 as indicated in table 1. Each cell [j][i] indicate weather task **i** is assigned to processor **j** or not. Having **x** in the table entry indicate the former case, otherwise it is the later. For example, cell[1][3] = x means task 3 is assigned to processor 1. Once table 1 is created, the fault tree can be constructed [10].

	T_1	T_2	T_3	T_4	T_5
$P_1(\lambda_1)$			X		
$P_2(\lambda_2)$	X				X
$P_3(\lambda_3)$		X			
$P_4(\lambda_4)$				X	
$P_5(\lambda_5)$					

Table 1 Fault Tree Table

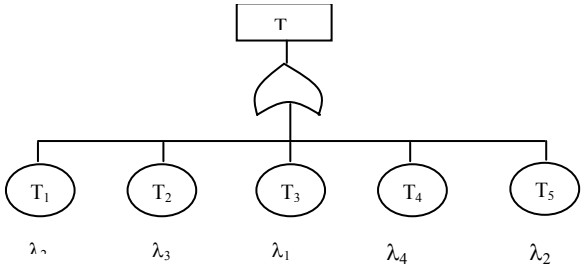


Fig 2 Fault Tree structure for (Fig 1)

In fig.2, Logically **OR** gate corresponds to a series system. Series reliability can be obtained from the following equation [10]:

$$R_S = \prod_{i=1}^{n} R_i \qquad (2)$$

where: R_i= The ith component Reliability.
R_s= the series system reliability.
n = the number of components.

When components failure times follows exponential failure laws becomes

$$R_s(t) = \exp-(\sum_{i=1}^{n} \lambda_i)t \qquad (3)$$

4 The New Scheduling Algorithm

In this section, we describe a scheduling method, which is based on introducing an objective function that combines the different scheduling goals. This objective function is used to guide the search algorithm to find a feasible solution.

4.1 Fitness objective function

The main goal of the scheduling problem is to allocate the fork-join structure to the processors of the system where the following are satisfied:

1- The deadline constraints
2- The reliability of the system is maximized.
3- The degree of parallelism is also maximized.

Each factor is represented by one term in the objective function. The first term reflects the meaning of real time applications. Real-time systems are defined as those systems in which the correctness and performance of the system depends not only on the logical results bust also on the time at which the results are produced. So the first term represents the effect meeting or violating the application deadline. The value of this term indicates how much the task deviates from its stated deadline. The second term represents the effect of processor reliability when we execute a certain task. The last factor indicates how much parallelism is achieved. The proposed objective function is represented by:

$$F = R_{dj} * R_{fi} * R_3 \qquad (4)$$

Now we explain the details of each of these terms:

- When an application is submitted, the participating processors are searched for processors that can run the tasks forming the application. The available processing power of the selected processors $\mathbf{PP_j}$ should be greater than the required processing power of the tasks **(t/d)**. The value **t** represents the total execution time of a task or a group of tasks on processor **j** and **d** is the total deadline of the same group of tasks.
  This term is calculated as follows:

$$R_{dj} = mis(t/d - PP_j) \qquad (5)$$

$$where: mis(x) = \begin{cases} 1-x & if \quad x > 0 \\ 1 & Otherwise \end{cases}$$

  As **x** increases the value of $\mathbf{R_{dj}}$ decreases and consequently the total objective function decreases. In other words, it measures how far the task is from achieving its deadline. The solution with large **x** is not desired.

- The second part of the overall goal is to assign the tasks to the most reliable processors and consequently increase the overall system reliability. Based on that , the second term represents the reliability of a processor when executing a task or a group of tasks on that processor. The reliability of this group of tasks is calculated from equation (1).

$$R_{fj} = R(n, j) \qquad (6)$$

The total reliability of the fork-join structure can e calculated from (3).

- The last factor reflects the effect of the parallelism on the overall schedule. Either the number of branches or the number of processors bound the value of this term.. R_{pj} can be represented as follows:

$$R_{pj} = \begin{cases} m & if \quad n > m \\ n & \quad m > n \end{cases} \qquad (7)$$

Where m = number of processors in the system
n = number of branches of the fork-join application

Since $\mathbf{R_f}$ varies in a small range, consequently the effect of the processor reliability change will be dominated by the group size effect. We use the natural logarithm to expand this range from **[α,1)** to **[β,0)** where β is a small negative value [11]. $\mathbf{R_f}$ will be modified to:

$$R_{fj} = -K / \ln R(n, j) \qquad (8)$$

The final objective function is as follows:

$$F(n, j) = (R_{dj}) * (R_{fj}) * R_j \qquad (9)$$

In order to obtain a feasible solution, this objective function should be maximized.

4.2 Scheduling Method (BFTG):

In this section we describe our scheduling algorithm, Best Fit using Objective Function (BFTG). The scheduling algorithm consists of the following steps:

i) Calculating the required processing power for each task in the application.

Required processing power for a specific task is defined as the ratio between the expected execution and the task target deadline [12]. For a single CPU, the value of the processing power is less than or equal to 1. Equation 10 indicates the required processing power ω_i of a task t_i is the ratio between the expected execution and the task target deadline d_i. To allow acceptable performance tolerance, safety factor σ may be added to the average execution time where σ represents the execution time variance.

$$\omega_i = \frac{E[\tau_i] + \sigma}{d_i} \qquad (10)$$

ii) Constructing the schedule table ζ

The scheduling table has two dimensions **(m×n)** where m (vertical dimension) is the number of processors and n (horizontal dimension) is the number of tasks for a certain application. In case of parallel structure, all tasks participating has the same priority to be scheduled. However, in constructing the scheduling table, the horizontal dimension elements (tasks) are sorted in deceasing order of its required processing power. In other words, we start with tasks that have larger required processing power. The content of each cell [i][j] of the table shows the objective function value calculated from equation (9).

iii) Finding a feasible solution

Given the scheduling table**ζ,** we describe an algorithm that finds a schedule with maximum accumulated objective function value Fig.3. Assume F is the overall objective function value for the application scheduling solution, which is initially set to 0. Starting from the first column, the algorithm explores the each column vertically searching for the cell with maximum value. Once it is found, Update the total objective function F and the Output Schedule. The procedure is continued for the next columns until the full schedule is achieved.

```
Input scheduling table ζ
Output Schedule [n];
Begin
   Obtain the scheduling table ζ
   F = 0;
   While there are tasks in ζ do
            i = 1;
            Start at column # i.
            Pick the entry with maximum value cell[j][i]
            Update F = F+ cell[j][i]
            Update Schedule [i] = j
            i = i+1
               End While
End
```

5 An Illustrative Example:

In this example, the distributed computer system consists of 8 nodes. Assume a fork-join Application with 8 Tasks. The first step is to calculate the required processing power for each task. Table (1) indicates the required processing power for different tasks for the application.

Task #	Task Time	Task Deadline	Rpp[i]
1	5,109	10,663	47.91
2	4,711	10,663	44.18
3	4,281	10,663	40.15
4	4,138	10,663	38.81
5	3,886	10,663	36.44
6	2,612	10,663	24.50
7	2,172	10,663	20.37
8	492	10,663	04.61

Table 1 required processing power for each task

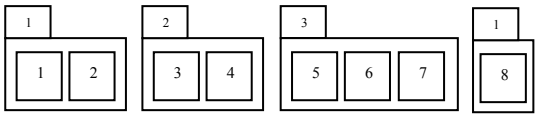
Second step, is to construct the scheduling table:

	T_1	T_2	T_3	T_4	T_5	T_6	T_7	T_8
P_1	264.4	266.5	269.1	270.0	271.7	282.7	288.0	266.5
P_2	257.3	259.4	261.9	262.8	264.4	275.2	280.3	325.2
P_3	257.3	259.4	261.9	262.8	264.4	275.2	280.3	325.2
P_4	248.8	250.8	253.2	254.1	255.7	266.0	271.0	314.4
P_5	249.3	251.3	253.7	254.6	256.2	266.6	271.5	315.0
P_6	252.4	254.5	256.9	257.8	259.4	269.9	275.0	319.0
P_7	256.8	258.9	261.4	262.2	263.9	274.6	279.7	324.5
P_8	256.6	258.7	261.2	262.1	263.7	274.4	279.5	324.3

Finally, by searching the scheduling table the schedule solution is:

Task No. : 01 02 03 04 05 06 07 08

Processor: 01 01 02 02 03 03 03 01



6 Simulation Results

In the simulation program, a randomly workload is generated and applied on a simulated multi-computer system. Throughout the simulation we assume the system consists of a fully connected 8 machines (processors). Each machine has a buffer that hold the tasks ready to be scheduled by that machine. The failure of the machines are assumed to be uniformly distributed between 0.001 and .0001 failure/hr.

The simulation studies proceed as follows:

1. Generate exponentially distributed applications (task graphs). The average execution time of each task is 3 sec.
2. The applications (task graph) arrived at the machine terminal are following Poisson distribution.
3. The inter arrival rates are ranged from 10 to 100 sec. We assume homogenous system. Therefore, that tasks execution times are the same on any machine. The total number of applications is 1000 per node.
4. Number of tasks in each application is uniformly distributed between (10 and 15) or (3 and 8).

Given the same workload and system parameters, three simulation studies are performed:

1. Apply Best-Fit for Task Graph (BFTG) algorithm.
2. Apply "First Match for Task Graph" FMTG. In (FMTG): application tasks are assigned to the first processor that satisfies the deadline required by the task.
3. Apply "First Match for Single Unit" FMSU. In FMSU, the application is treated as single unit task and it is assigned to the first processor that matches the deadline requirements of the application.

To evaluate these studies, three metrics are used. The first metric is the acceptance rate measure. The acceptance rate is defined as the ratio between the number of accepted application by all machines and the total number of applications arrived on all the machines input buffers. The second metric is the system reliability. System reliability is defined as the probability that the system can execute tasks without failure. The last one is the average parallelism degree. Parallelism degree is defined as the max number of branches (or processors) used in the scheduling method.

Fig.3 shows the simulation study that indicates the effect of the proposed scheduling method on the system acceptance rate. The results shown is the average of the data obtained in 1000 experiments per node. According to the simulation results, the performance of BFTG is better than both FMSU and FMTG in different cases. These cases are based on the amount of violation allowed by the system. For example in case of no violation allowed, BFTG outperform the others. As a logic result, BFTG will outperform the other algorithms in case of allowing a deadline violation. From that we can conclude that, by controlling the parameters of the objective function, BFTG can be suitable for either had real-time or soft real time with an outstanding performance.

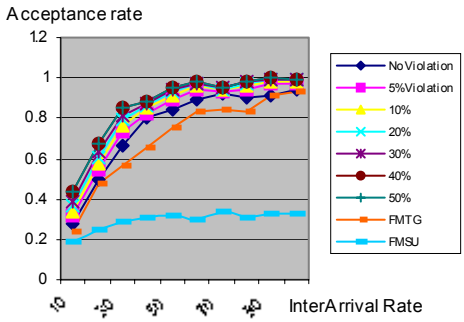


Fig.3: The acceptance rate performance

In Fig.4, we measure that the reliability performance when we consider only applications accepted by the three algorithms. According to the simulation results, the average performance of BFTG in all cases is better than FMTG and FMSU. This is due to the fact that, assigning the whole application to the same processor means that the processor will run the application for longer time which increase the probability of having a failure. In case of task graph applications, the situation is different. Small tasks are assigned to processors and consequently the processor will run for shorter period of times and that decreases its probability of failure.

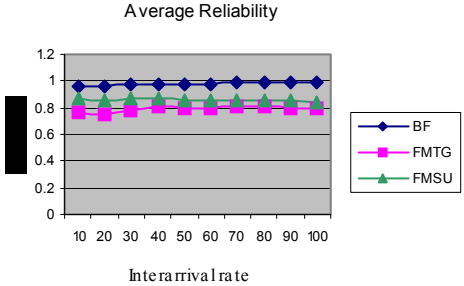


Fig.4: The reliability performance

In Fig.5 we measure that the reliability performance when we consider only applications accepted by both BFTG and FMTG. Also, in this case BFTG outperforms FMTG.

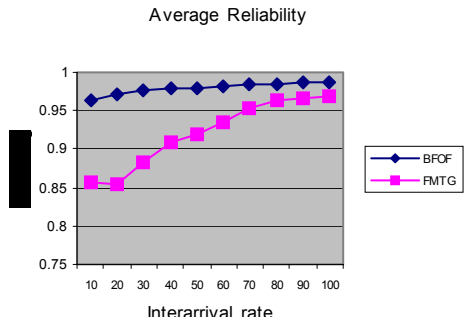


Fig.5: The reliability performance for BFTG and FMTG

Fig.6 shows the third simulation study that investigates the effect of the proposed scheduling method on the parallelism degree. According to the simulation results, the performance of BFTG is better than FMTG and FMSU. This result indicates that having the parallelism term in the objective function leads to an improvement in the average parallelism degree achieved by the scheduler.

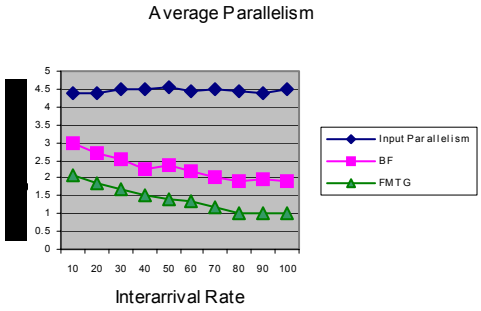
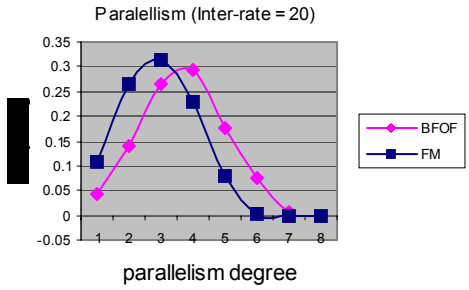


Fig.6: The average parallelism performance

Also, Fig.7 shows the relation between the parallelism degree and the frequency of having that degree for a specific inter-arrival rate. The study shows that BFTG outperforms FMTG (FMSU is excluded since it has no parallelism at all).



As a general conclusion even though the acceptance rate of our algorithm slightly outperforms the FMTG, we validate another parameters at the same time such as reliability and degree of parallelism.

7 Conclusions

In this paper, we presented a new algorithm for scheduling parallel structures of a real-time application into a cluster of processors with possible failures. We developed an objective function to guide the searching process to find the best task assignment that simultaneously achieve the required deadline and maximize both of the reliability and the degree of parallelism. Simulation results indicate that the new algorithm generates a schedule with much better performance.

References

[1] K. G. S. a. P. Ramanathan, "Real-Time Computing: A New Discipline of Computer Science and Engineering," Proc. IEEE, vol. 82, pp. 6-24, 1994.

[2] N. K. J. S. Srinivasan, "Safety and Reliability Driven Task Allocation in Distributed Systems," IEEE Transaction on Parallel and Distributed Systems, vol. 10, 1999.

[3] F. O. Atakan Dogan, "Matching and Scheduling Algorithms for Minimizing Execution Time and Failure Probability of Applications in Heterogenous Computing," TEE Transactions on Parallel and Distributed Systems, vol. 13, 2002.

[4] H. J. Xiao Qin, C.S. Xie, and Z.F. Han, "Reliability-driven scheduling for real-time tasks with precedence constraints in heterogeneous distributed systems," Proc. of the 12 th International Conference Parallel and Distributed Computing and Systems, pp. 617-623, 2000.

[5] G. M. a. C. S. R. Murthy, "A Fault-Tolerant Dynamic Scheduling Algorithm for Multiprocessor Real-Time Systems and Its Analysis," IEEE Transactions On PARALLEL AND DISTRIBUTED SYSTEMS, vol. 9, 1998.

[6] R. M. a. D. M. S. Ghosh, "Fault Tolerance Through Scheduling of aperiodic Tasks in Hard Real-Time Multiprocessor Systems," IEEE Trans. On Parallel and Distributed Systems, vol. 8, pp. 272-284, 1997.

[7] Y. K. a. T. K. Tatsuhiro Tsuchiya, "Fault-Tolerant Scheduling Algorithm for Distributed Real-Time Systems," WPDRTS '95, 1995.

[8] I. P. Pascal Chevochot, "An Aproach for Fault-Tolerance in Hard Real-Time Distributed Systems," SRDS, 1998.

[9] J. W. a. M. G. S.M. Shatz, "Task Allocation for Maximizing Reliability of Distributed Computer Systems," IEEE Transaction on Computers, vol. 41, 1992.

[10] E.E.Lewis, "Introduction to Reliability Engineering," John Wiley & Sons, 1987.

[11] S. W. a. S. Gokhale, "Exploring Cost and analysis Tradeoffs in Architectural Alternatives using Genetic Algorithms," ISSRE 99, pp. 104-113, 1999.

[12] A. A. Alhamdan, "Scheduling Methodss for Efficient Utilization of Cluster Computing Enviroments," PhD. Thesis, 2003.