Military Technical College
Kobry El-Kobbah,
Cairo, Egypt

8th International Conference
on Electrical Engineering
ICEENG 2012

Fault-tolerant scalable hierarchical scheduling in grid computing

By

Gamal A. El-Sayed* Aref M. Abdullah**

Abstract:

Computational grids have the potential for solving large-scale scientific applications using heterogeneous, distributed and possibly non-dedicated resources. Grid environment is dynamic in nature, hence scalable and fault-tolerant scheduling is a much needed to schedule parallel applications with inter-process communication. In this paper, we propose a hierarchical and fault-tolerant scheduling approach, in which the application's processes communicate indirectly by sending messages over the network through mailbox-based communication technique at a shared node. In grid, process often migrates from one node to another, so this technique ensures the reliable delivery of messages; prevents messages sent to the migrating process form losing. A non-evolutionary mapping heuristic based on Max-Min approach is also proposed for mapping such applications on grid resources. Finally, MPICH-V1 protocol is integrated into our scheduling framework that exploits the mailbox-based technique instead of channel memories. The simulation experimental results demonstrate that, the proposed approach as a whole effectively schedules the grid applications in scalable and fault tolerant way thereby ensures the application to be executed within its deadline making the grid environment trust worthy.

Keywords:

Grid **s**cheduling, fault-tolerance, MPICH-V1, rescheduling**,** checkpointing.

---

* *Electrical Engineering Department, Assiut University,* Egypt
** *Electrical Engineering Department, Assiut University,* Egypt

1. Introduction:

Grid computing has emerged as a new paradigm for distributed computing, having developed as a mechanism for allowing collections of connected computer systems to form large-scale data and computing networks. It promotes the sharing of distributed resources that may be heterogeneous in nature, so as to enable different application domains including science, industry, engineering, finance and even government to solve large-scale computing problems [1]. To achieve the promising potentials of computational grids, an effective scheduling system is fundamentally important. A grid becomes useful and meaningful when it both encompasses a large set of resources and serves a sizable community. In grid, the abilities of the computing resources are heterogonous and non-dedicated, and applications\jobs often arrive dynamically. Because of these, scheduling methods of parallel computing may not be applicable in a grid. Through good scheduling methods, the system can get better performance, as well as applications can avoid unnecessary delays [2].
Scheduling on a grid has three main phases. Phase one is resource discovery, which provides a list of available resources. Phase two is resource allocation, which involves selection of feasible resources and mapping of applications to resources. The third phase includes application execution, which includes file staging and cleanup. In second phase, the selection of the best match of applications to resources is an NP -complete problem [3]. Existing scheduling mechanisms can be classified into two categories: on-line mode and batch mode. Using an on-line mode technique, the application is scheduled to resources as soon as it arrives. In batch mode, however, arriving applications are collected for examination at prescheduled times and these applications are scheduled as a meta-task when a scheduling event is triggered.

Parallel application scheduling is a kind of allocating problem consisting of allocating machine resources to applications. The allocating can be based on either space sharing or time sharing of computational resources. With space sharing, machines are partitioned into groups, with each group assigned exclusively to one parallel application. Most application schedulers are based on the space-sharing approach [4][5][6]. Under such model, high performance schedulers act well. However, resource utilization and execution efficiency cannot be guaranteed if applications and machines are not homogeneous. For a practical grid platform with high heterogeneity, the space sharing approach with a simple model is not an attractive choice [7].
With time sharing, each machine can run processes of multiple applications concurrently. The basic benefits of time-shared execution are high system utilization and short average application response time. However, developing high performance scheduler in time sharing model is more difficult than in space sharing especially for parallel applications with inter-process communication, but it can be achieved by carefully planning and adapting resource allocation to applications [7].

In this paper, we focus on online time sharing scheduling of compute-intensive parallel applications with inter-process communication which their constituted processes may send or receive messages at random point on non dedicated heterogeneous grid. The objective of our scheduler is guaranteeing low execution time of application as well as satisfying user-dependent requirements by introduce fault-tolerant scalable scheduling framework.

For supporting time-sharing-based parallel application scheduling, previous researches mainly focused on co scheduling mechanisms supporting time-shared execution, such as gang scheduling and many loosely coordinated scheduling mechanisms [8][9][10][11]. They are concerned with mechanisms to schedule processes distributed across machines at the same time through coordination control. However co scheduling mechanisms do not tackle the problems caused by application diversity and resource heterogeneity in a grid.

Due to the existence of a large number of resources and users in grid environment, issues such as autonomy and heterogeneity further complicate the allocation problem. In such an environment, the feasibility, efficiency and scalability issues must be taken into account, so it is not realistic that using a centralized scheduling scheme to deal with all the resources available in grid environment [12]. Furthermore, the probability of a failure is higher in the grid computing than in a traditional parallel computing and the failure of resources affects application execution fatally. All of these push towards the need for scalable and fault tolerant scheduling system [13]. Therefore, efficient scheduling in scalable and fault tolerant manner for time-sharing execution is a much needed functionality that is absent in today's grid resource management infrastructure especially for parallel applications with inter-processes communication which their constituted processes may send or receive messages at random point.

A large number of research efforts have already been devoted to fault tolerance. This work can be divided into pro-active and post-active mechanisms. In pro-active mechanisms, the failure consideration for the grid is made during allocation phase and before the dispatching of the application to selected resources, so it is dispatched with hopes that the application does not fail. Whereas, Post-active mechanisms handles the application failures after it has occurred. Many works are primarily post-active in nature and deal with failures through Grid monitoring.

In addition, time-sharing execution brings up another problem. Parallel application scheduling in such high volatile and fault-prone environment is a dynamic procedure. So, rescheduling the affected processes in case of failure is thus a desired functionality. This is often implemented through checkpointing in combination with migration.

In our work, we follow the best-effort policy in which local jobs have higher priority than a grid task. Also, our proposed scheduling system will be scalable and fault tolerant aims to guarantee the required QOS of the grid application. We use Post-active fault tolerant mechanisms to recover and migrate the affected processes from the faulty node

to another new node. Some of the main contributions of the study include:

- New mailbox-based communication model is suggested which is suitable for highly dynamic environment where processes frequently migrate from one node to another.
- We adopt a four-level scheduling approach which divides the big scheduling problem into sub-problems.
- Due to using the mailbox-based communication model, we proposed non-evolutionary communication-aware heuristic mapping based on Max-Min approach for mapping parallel application with inter-process communication.
- MPICH-V1 protocol is slightly modified and integrated in our approach that exploits the proposed mailboxes space as events logger, its fault tolerance functionalities is used to checkpoint and to migrate the affected processes from the failed node the another available resources.
- Finally, our proposed approach is compared with the work proposed by [14] .

In this paper, we assume that the grid middleware layer supplies the appropriate services for co-allocation [15] and resource management. Also, our suggested work relies on some existing package services for application and resource monitoring.

This paper is organized as follows. Section 2 briefly reviews related work. In Section 3, we present our modeling framework, the proposed scheduling method, and also the integration of MPICH-V1 into our scheduling system, and then in Section 4, we present our simulation experiments and show the comparison results. Finally, the conclusion is presented in section 5.

2. Related Work:

Due to the practical relevance of grid task allocation and task scheduling, several approaches have been investigated; some of them are very effective for specific classes of problems and the other work well only homogenous platforms, as for instant [16] [2][17][18] deal with independent task workloads, loosely dependent task mapping [19], single site mapping [20][6][21], or multisite mapping [4][22][23][24]. Furthermore, evolutionary algorithms have been explored [14][6] to find a near-optimal solution. However, these approaches are limited to small size instances due to the time-consuming nature of them, and they are not the most suitable for online mapping and remapping in large scale grid.

In recent years, the researchers have proposed several scheduling methods that are using in computing to allocate grid resources for parallel applications. The majority of these studies were based on centralized scheme [25][26][2][27][28][29][18] while there is little works based on decentralized [30] and hierarchical schemes [31][32][21] [5][33].

Some efforts on application-level scheduling focused on applications represented by DAG graphs [34], by task interaction graphs [35][14]. Building task graphs expressing precedence constraints for complex parallel applications is non-trivial and hence cannot be adopted for a large number of parallel applications. Some efforts use performance models that predict execution times for parallel applications [36][27][20][37][14] .These performance models are often used by optimization algorithms that search the space of candidate schedules and choose the schedule with the minimum predicted execution time[20] [14].

Application Level Scheduling (AppLeS) was proposed in [38][36] for adaptive application scheduling on heterogeneous computing platforms. The AppLeS approach exploits static and dynamic resource information, performance predictions, application-specific and user-specific information, and scheduling techniques to adapt application execution. The scheduling algorithms in AppLeS rely on short-term prediction provided by the Network Weather Service. AppLeS uses a performance to determine a set of candidate schedules for the target application and then chooses the best schedule that matches performance criteria or gives minimum makespan is selected. Currently available templates are APST for parameter sweep application (APST), AMWAT for master/worker applications and SA for scheduling moldable applications on space-shared parallel computers. However, AppLeS doesn't address the scalability and fault tolerance issues. Also, there is potential for schedule conflict that may arise between different schedules when simultaneous requests submitted to different AppLeS agents.

A project presented by [35] is targeted for online scheduling for parallel applications follows a single program multiple data (SPMD) style of program execution. It used the interacting graph formulation method to formulate the scheduling problem. In this type of graph, the nodes represent processes and the edges represent intensity of interaction between those processes. Its grid consists of several zones; each zone has a scheduler called the zone scheduler. This scheduler is responsible for making the scheduling decision for all resources in that zone, which it is responsible for clustering the application processes to identify subsets (or "clusters") of processes that communicate a lot among themselves.. Thus, the tasks of an application are split into clusters, with processes in each cluster interacting heavily and relatively infrequently with processes in other clusters. After clustering the application's processes, the zone scheduler that receives the application sends out requests to its schedulers of other zones to schedule one or more clusters .Each zone is responsible for making its own scheduling decision. Like AppLeS, there is potential for schedule conflict that may arise between different schedules when simultaneous requests submitted to different zone scheduler, that because their scheduling approach suffers from a crucial problem which is lack of a global vision and coordination between the schedulers of the different zones.

In [14], an adaptive multisite mapping for computationally intensive grid applications based on multi-objective differential evolution was proposed by *Ivanoe et al.* to

automatically provide a set of possible mapping solutions for applications that constituted of communicating tasks in non-dedicated and heterogeneous grid. It gave different balance for resources use and QoS constraints, investigated only in terms of the degree of reliability of the grid nodes and of the internet links of the sites they belong to. It is consider fault tolerance in pro-active manner during mapping and exclude those resources that have tendency to failure. But using the post-active fault tolerance mechanism sometimes outperform using pro-active one because runtime rescheduling is a desired functionality due to dynamic and fault prone nature of grid, however their effectiveness largely depends on tuning runtime parameters such as the checkpointing interval and which processes should take checkpoint so as to limit its overhead. Also [14] doesn't consider the growth in size of grid since it followed a centralized scheme approach.

3. Scheduling framework:

3.1 Proposed architecture:

In our grid model, the resources are connected via 4-level hierarchical network. The proposed grid consists of several zones each with many sites. The site in a zone consists of several heterogeneous workstations in time sharing mode which that can be grouped according to their physical locations. For simplicity, we assume that each workstation with single processer. The workstations in each site are connected by LAN network and managed by a node called a site master node. All sites in a zone may be connected by LAN, WAN or by internet and managed by a node called a zone master node. The different zones are also linked through the internet and managed by a node called a super grid master node. For this paper, we assume that all master nodes are dedicated nodes with high outgoing/ingoing bandwidth; also we assume that these master nodes don't fail.

At the top of our hierarchical model tree, there is a Super Grid Scheduler (*SGS*) in the super grid master node to which the users submit their applications, in level 2 there are zone schedulers *(ZS_l)* in zone master node, sites schedulers (SS_i) in level 1, and at the level 0 home schedulers run on each computing resource (the individual processor) in time sharing mode.

In our model, Processes of a particular application on different processors only communicate indirectly by sending messages over the network through mailboxes which are dynamically created in mailbox pool which resides at a site, zone or super grid master nodes before dispatching the application to the selected resources as illustrated below in subsection 3.2.

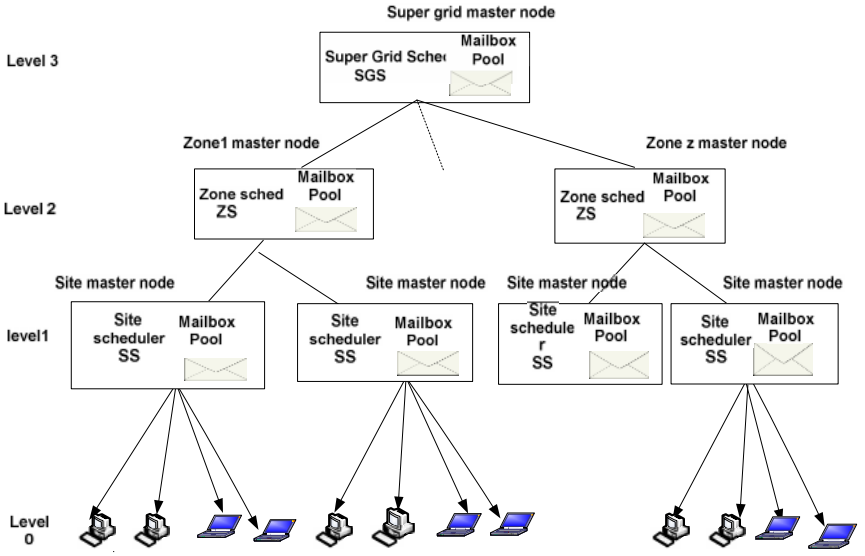


Figure (1): *The hierarchical scheduling architecture*

3.2 Our mailbox-based communication scheme:

In our mailbox-based communication scheme, each process of the parallel application is assigned a mailbox queue by the scheduler at the master node which all selected resources for application are directly located under its control before dispatching it to the selected resource to buffers the messages sent to it, i.e. the mailboxes queues for all processes of the application are located in mailbox pool which residing at the master node which is different from the nodes where those processes will be executed. For example, if the selected resources for all processes of an application are located in a single site, a mailbox for each process is created in the site master node of that site before it is dispatched by the site scheduler to the selected resource.

Inter-process communication between processes can occur as follow:

• Transmission of a message from the sender to the receiver's mailbox: If a process P_i wants to send a message m_i to a process P_j, it sends it to P_j's mailbox at the master node.

• Delivery of the message from the mailbox to its owner process: The receiving process can use a push or pull operation to obtain the message from the mailbox as follows:

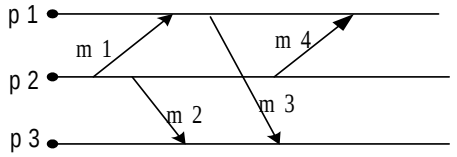
Push (PS): The mailbox controller keeps its owner process's address and forwards every message to it. Although message queries incur no costs, the scheduler must notify the mailbox of the current location of its owner process after scheduling or rescheduling process. PS mode is needed if real-time message delivery is required.

Pull (PL): The process retrieves messages from its mailbox queue (in order) whenever needed. The mailbox doesn't need to know the process's current location, thus avoiding location registration, but the process must query its mailbox for messages. Although

polling messages would increase message delivery overhead, it ensures reliable message delivery and very useful technique in migration.

In this paper, we will adopt a pull operation method. When a process wants to retrieve a message m_i from its mailbox, it uses a primitive like get (Addr, mailbox [y]) to pull the one at the location index y of its mailbox queue, where Addr refers to the full address of the mailbox.

Figure 2 show s a part of an application's execution consists of three processes and the corresponding mailboxes' view



(a)

Mailbox of P_1			m_4	m_1
Mailbox of P_2				
Mailbox of P_3			m3	m2

(b)

***Figure (2)**: (a) A part of an application's execution (b) Mailboxes pool view.*

In highly volatile and non-dedicated grid environment in which the nodes can leave or crash at any time, the application often require to migrate from one node to another. Therefore, in our mailbox-based scheme the messages sent to the faulty process during its migration will not be lost and they don't need to be routed or retransmit. They will be kept in its mailbox and the process will pull them after restarting on the new resource.

Table 1 shows the definition of notations used in this paper.

***Table (1):** Definition of notation*

Notation	Meaning
R_{sg}^{avail}	The total no. of available computing resources in entire grid at time of scheduling the job (application).
R_{sg}^{sat}	The resources set in entire grid that satisfies the minimum requirement of a job.
Rz_l^{all}	The total no. of available computing resources in zone z_l at time t.
Rz_l^{sat}	The resources set in a zone z_l that satisfies the minimum requirement of job.
Rs_i^{sat}	The resources set in site s_i that satisfies the minimum requirement of a job
R_G^{sat}	The resource set that satisfies the minimum requirements of job in

	general speaking (in a zone, a site or in the whole grid accordingly.
r_j^{speed}	The CPU speed of computing resource j in MIPS.
$r_j^{\%}$(t)	The predicted percentage of available CPU for the resource j at time t
P_i^{size}	The no. of instructions of process P_i measured in MI (mega instructions)
P_i^{cmm}	The summation size of all messages expected to be sent or received by process p_i measured in bytes and extracted from previous run.
SMN	Site master node.
ZMN	Zone master node.
$SGMN$	Super grid master node.
$avaibw_j^{SM}(t)$	The predicted available bandwidth between resource j and the manager node of site s_i at time t.
$avaibw_j^{ZM}(t)$	The predicted available bandwidth between resource j and the manager node of zone z_l at time t.
$avaibw_j^{SGM}(t)$	The predicted available bandwidth between resource j and the super grid manager node of the system at time t.
$avaibw_j^{MN}$	The predicted available bandwidth between resource j and a manager node in general speaking at time t.
$T_{exe(i,j)}$	The estimated computation time of p_i on processor j based on future resource state predicted by NWS.
$T_{tot(i,j)}$	The total time required by process p_i to finish its work on resource j.
$T_{comm(i,j)}$	The total time which will be spent by p_i executing on resource j to transmit all its messages from its mailbox in case of receive events or to mailboxes of other processes in case of send events.
$T_{trans(i,j)}$	The estimated transmission time of the process p_i from the scheduler to the resource j.

3.3 Performance Model:

The mode of MPI communication in our approach is assumed to be either asynchronous send or blocking receive. So instead of spin-block [11], we assume that block-immediately mechanism is adopted for the process blocking on receiving messages if the communication daemon of the process checked its mailbox and found that the wanted message had not arrived yet. The process voluntarily relinquishes the processor so that next competing process can be scheduled, and when the required message arrives, it re-acquires the processor for time proportional to the portion of its time slice it relinquishes to the other process.

In this work, we assume to know, for each process P_i, the number of instructions P_i^{size} and the total amount of communications P_i^{cmm} a process may exchange with all

other processes. Also, information on the status of available resources is also very important for a grid scheduling to make a proper schedule, especially when the heterogeneous and dynamic nature of the grid is taken into account. The role of the grid information service (GIS) is to provide such information to grid schedulers. For example, in the Globus Toolkit [39], which is a standard grid middleware, most schedulers fetch predicted resource parameters from GIS which is responsible for collecting and predicting resource state information, such as CPU capacities, memory sizes, network bandwidth and load of a site in a particular period. The Globus Monitoring and Discovery System (MDS), which is normally maintained as a hierarchical structure, and Network Weather Service (NWS) [40] are examples of GIS. Thus it is clear that there exist different software components to retrieve run-time resource information. Here, this information is supposed to be acquired either through statistical estimations in a particular time span or gathered by tracking periodically and forecasting dynamically resource conditions [40, 42].

Due to the hierarchical organization and the communication scheme in our approach in which the application's processes communicate through mailboxes reside in a shared node (a master node), there is no need for measuring the bandwidths between computing recourses and reporting them to MDS. We need only to measure the bandwidth between each computing resource and its parent and grandpa master nodes. This will minimize the overhead of sending dynamic information of bandwidth to MDS.

Considering the nature of our communication scheme in which the scheduler creates a mailbox for each process of the application before dispatching it to the selected resource, the communication time required by a process for pulling the messages from its mailbox and for sending messages to the mailboxes of other processes must be considered when predicting the expected completion time of the process. Thus the expected completion time of the process p_i on processer j is naturally modeled as the sum of estimated computation time, estimated communication time and estimated transfer time:

$$T_{tot(i,j)} = T_{exe(i,j)} + T_{comm(i,j)} + T_{trans(i,j)} \qquad (1)$$

Where:

-$T_{exe(i,j)}$ is the estimated computation time of p_i on processor j with its future state predicted by NWS, which is calculated as $T_{exe(i,j)} = \frac{P_i \text{size}}{r_j \text{speed} \times r_j\%}$.

- $T_{comm(i,j)}$ is the estimated total time which will be spent by p_i executing on resource j to transmit its all messages from or to mailbox pool which resides at a master node MN (SMN,GMN, or SGMN). $T_{comm(i,j)}$ is determined as :

$$T_{comm(i,j)} = \frac{P_i cmm}{avaibw_j \text{MN}} \quad .$$

-The estimated transmission time of the process p_i from the scheduler to the resource j can be determined as:

$$T_{trans(i,j)} = \frac{P_i \text{size}}{avaibw_j \text{Mn}} .$$

Using the above performance model to predict the estimated completion time of each process including the computation time, communication time, and transfer time the scheduler can choose the best set of resources to optimize the application QoS based on the information provided by performance modeling. Whereas the total completion time of the application is decided by the process has the maximum completion time.

3.4 scheduling policy:

While workflow and less communicating parallel applications achieve good performance when executed across multiple sites, compute-intensive parallel applications which involve significant inter-task communication exhibit poor performance due to low-speed network links between the sites. So, the communication cost is a significant factor to be considered during resource allocation for such parallel applications to achieve better performance in large-scale grid.
Here, we will first try to find the best resource set (schedule) to execute the application inside a single zone. But if there is no a single zone with insufficient resources which satisfy the user-dependent requirement, our scheduling system will then try to find resources from multiple zones as follow:

1- When an application scheduling request is submitted by a user to the super grid scheduler (*SGS*), the *SGS* quires the grid information system to check the available resources, and then redirects the request to each of the zones schedulers in lower level which has enough resources to execute the application under its control. Then SGS will wait for each of the zone schedulers to which it has redirected the scheduling request to calculate the different possible schedules and then return its best schedule to it. When the *SGS* receives the offered schedule of each zone scheduler, it compares them and chooses the best one to execute the application (the best one is the schedule with minimum completion time).

Once a zone scheduler ZS_l receives a scheduling request from *SGS* scheduler, it first gets a set of possible resources which the meet the minimum requirement for the application and if the candidate resources are enough to execute the application, then the best schedule inside that zone will be found as shown in the following. If not, the zone scheduler ZS_l returns the *NULL* schedule to the super grid scheduler *SGS*:

a. Once a zone scheduler ZS_l receives the scheduling request, it must first check whether there is any single site under its control with enough candidate resources as requested by application. If so, the ZS_l scheduler will get the dynamic

information of these candidate resources and the available bandwidth between them and that site master node *SMN*, and then calculates the best solution that can be offered by this site by applying the mapping algorithm we present in the next subsection. After the ZS_l has calculated the best schedule of each of the candidate sites, it compares them chooses the best one (the schedule of the site that gives minimum completion time).

b. The zone scheduler ZS_l also calculates the best schedule of all available resources under its control (combining resources from multiple sites) by getting the dynamic information of these candidate resources and the available bandwidth between them and the zone master node *ZMN* and then applying the proposed mapping algorithm.

c. Finally, the zone scheduler ZS_l compares between the best solution of the best site and the best solution of the whole zone and returns the best one back to the *SGS* scheduler.

2- If there is no zone scheduler with enough resources to execute the application, then the super grid scheduler (*SGS*) calculates the best schedule among all available resources under its control (by selecting best resources across sites under the control of different zone schedulers) by getting the dynamic information of the set of possible resources which meets the minimum requirement for the application the dynamic information of these candidate resources and the available bandwidth between them and the super grid master node *SGMN* and then applying the proposed mapping algorithm. Finally, After the *SGS* scheduler chooses the final schedule either inside single site, single zone, or multi zone, the actual application is dispatched to the selected resources as follow:

i. If all the selected resources for the application are located in a single site, the application itself is submitted to the site scheduler which is responsible for creating a mailbox for each of the application's processes in the mailbox pool at the site master node *SMN* and then dispatches each process to its selected resource. In this case, the site scheduler is also responsible for monitoring and rescheduling the application in the presence of resource\link failure.

ii. If all the selected resources are located in more than one site under the control of a single zone scheduler, then the application itself is submitted to that zone scheduler which, in this situation, will be responsible for creating a mailbox for each of the application's processes in the zone manager node *ZMN* and then dispatches each process to its selected resource. In this case, the zone scheduler is also responsible for monitoring and rescheduling the application in the presence of resource\link failure.

iii. If all the selected resources are not located under the control of a single grid scheduler, then the application itself is submitted to that *SGS* scheduler which, in this situation, will be responsible for creating a mailbox for each of the application's processes in the mailbox pool in the supper grid master node *SGMN*, and then dispatches each process to its selected resource. In this case, the *SGS* scheduler is also responsible for monitoring and rescheduling the application in the presence of resource\link failure.

3.5 The proposed Mapping Algorithm:

Due to using the proposed mailbox-based communication model, it can be now easy to modify and use one of those simple mapping heuristics which normally used in mapping independent meta-task for mapping inter-process parallel application which their constituted processes may send or receive messages at random point on non-dedicated and heterogonous grid.

Indeed, using a pull operation by processes to get their messages from their mailboxes, at a shared node, slightly increases the execution time of application. But for the non-dedicated and fault-prone environment like grid, where processes often migrate from one node to another, this technique ensures the reliable delivery of messages and prevents messages sent to the migrating progress form losing and retransmitting. Also the fixed-location mailbox used by our approach eliminates the need for message routing mechanism, and thus improves the performance.

Here, we will slightly modify the Max-Min heuristic by considering the communication cost a process needs to pull messages from its mailbox and to send messages to the mailboxes of other processes.

As explained in subsection 3.4, our mapping algorithm is invoked by super grid Scheduler (*SGS*) or a zone scheduler ZS_l after resource discovery phase. After getting a set of possible resources R_G^{sat} in which the user, who submitted the application, has an account and they meet the minimum requirement of the application (within a site, a zone or multi zone) which are obtained during resource discovery phase, the scheduler gets the predicted dynamic information of those computing resources and the predicted available bandwidths between them and the master node GMN from NWS and then invokes the algorithm mapping. The algorithm mapping uses the resources and application information as input and acts as follow:

Note: Let R_G^{sat} used to refer to the resource set (in general speaking) which satisfies the minimum requirement of application either in a zone Rz_i^{sat}, or in a site Rs_i^{sat} or in the whole grid R_{sg}^{sat} accordingly.

First, the mapper uses the performance model explained in subsection 3.3 (formula 1) to predict the expected completion time $T_{tot(i,j)}$ for each p_i of the application on each resource j □ R_G^{sat}. Remember that the communication time $T_{comm(i,j)}$ is the time required by p_i executing on resource j to transmit its all messages from its mailbox in case of receive events or to mailboxes of other processes in case of send events.

After computing the expected completion time of each process over all the computing resources, the Max-min approach will be used to map the processes of the application. It chooses the process having maximum of earliest completion times and mapping it to the corresponding machine. Then, the mapped process is marked and the selected recourse is removed from the candidate resource set R_G^{sat}. Then the procedure is repeated until all the processes are mapped. The figure 3 shows the pseudo codes of mapping algorithm.

1- let R_G^{sat} = a set of n possible resources in which the user submitted the job has as account in and meet the minimum requirement for the job
2: let *Jb* = a job with *N* processes { p_1, p_2,…… p_N}
3: let $T_{tot(i,j)}$= expected completion time of p_i on computing resource *j*.
4: for all p_i in the job *Jb DO*
5: for all *j* □ R_G^{sat} *Do*
6: calculate $T_{tot(i,j)}$
7: end for
8: end for
9: determine the minimum completion time of each p_i □ *Jb;*
10: map the $p_i$which has the maximum of the minimum completion time to corresponding resource *j*;
11: remove the p_i and *j* to avoid choosing it for other processes of the job *Jb*
12: determine the next maximum of minimum completion time among all the remaining processes after excluding the selected resources for mapped processes;
13: repeat steps 10-12 until all p_i □ *Jb* are mapped
14: *Bestsched* = mapping result
15: return *Bestsched* to caller
End

Figure (3):*The pseudo code of mapping algorithm*

3.6 Fault tolerance and MPICH-V1:

Due to the high rate of failures and the dynamic nature of the grids, the integration of fault tolerance with scheduling is very important. Also, the larger size and the high volatile nature of grid make failures accurse often during the application execution in an unpredictable way, thus the performance achieving integrating the post-active fault tolerance into the scheduling system through checkpointing and migration mechanism often outperforms using pro-active one.
Two main techniques are used for saving the distributed execution state and recovering from systems failures: coordinated checkpoint and uncoordinated checkpoint associated with message logging. In all cases, the process states are saved in reliable media.

In coordinated checkpoint processes coordinate their checkpoints in order to save a system-wide consistent state. It involves the rollback of all processes from the last checkpoint when a faulty situation is detected, even when a single process crashes. Uncoordinated checkpoint protocols allow all processes to execute a checkpoint independently of the others. Thus this technique relies on message logging in addition to process checkpointing to ensure the complete description of a process execution state in case of its failure. Message logging protocols consist in 1) logging all received messages and 2) re-sending the same relevant messages, in the same order, to the crashed processes during their re-execution. This principle provides the guaranty that a re-executed process starting from a previous correct state (the beginning of the execution or a consistent checkpoint image) will reach a state matching the rest of the system, as before the crash. There are three kinds of message logging protocols that can be combined with uncoordinated checkpoint [41][42]:

- Optimistic log (in which we can include the sender- based techniques) assumes that messages are logged, but that part of the log can be lost when a fault occurs. Either this technique uses a global coherent checkpoint to rollback the entire application when too much information has been lost, or they assume a small number of faults (mostly one) at one time in the system;
- Causal log is an optimistic log, checking and building an event dependence graph to ensure that potential incoherence in the checkpoint will not appear;
- Pessimistic log a transaction logging ensuring that no incoherent state can be reached starting from a local checkpoint of processes, even with an unbounded number of faults.

There have been many MPI implementations, which provide fault-tolerance to the applications. Different implementations differ in the way they store the state of execution and restore it back, the level in the software stack at which they are implemented and the level of the transparency to the user. Some such implementations

are MPICH-V1, MPICH-V2, FT-MPI, CoCheck, StarFish and Clip. These implementations have normally concentrated on how to tackle node disconnections, which may be due to node crash or network link failure. Fault tolerance mechanisms used by the above implementations are checkpointing, message logging, and the replication of processes [43]. In this paper we have slightly modified and integrated the MPICH-V1 with our scheduling approach. MPICH-V1[44] is the first protocol of MPICH-V designed to tolerate a high node volatility. It is based on uncoordinated checkpointing and pessimistic message logging protocol storing all communications of the system on reliable media. MPICH-V1 runtime consists of several entities: Channel memories, Dispatcher, Checkpoint servers, and computing nodes. To ensure this property, every computing process is associated with a reliable process called Channel Memory. Every communication sent to a process is stored and ordered on its associated Channel Memory. To receive a message, a process sends a request to its associated Channel Memory. After a crash, a re-executing process retrieves all lost receptions in the correct order by requesting them to its Channel Memory. A main property of MPICH- V1 is the uncoordinated restart: a process re-execution is independent of the other processes of the system [45].

There can be one or more checkpoint servers depending on the size of the MPI application. The checkpoint server collects the checkpoint images from the communication daemons at computing resources and provides them with the checkpoint image on restart. The dispatcher in MPICH-V1 manages tasks that are instances of traditional MPI processes. During the initialization, the dispatcher launches a set of tasks on participating nodes, each task instantiating a MPI process. A key task of the dispatcher is the coordination of the resources needed to execute a parallel application\job. This includes providing nodes for executing services (Channel Memories and Checkpoint Servers) and MPI processes. The dispatcher also monitors the participating nodes and detects a potential failure: either a node or a network failure. It then launches another task (a new instance of the same MPI process). The task restarts the execution, reaches the point of failure and continues the execution from this point.

2.7 Integration MPICH-V1 with our scheduling approach

The distributed nature of MPICH-V1 makes it easy to integrate it with our hierarchical scheduling framework. For integration our scheduler with MPICH-V1, we have used the MPICH-V1 dispatcher component as a dispatcher of our scheduler and also used the mailboxes pool instead of channel memories. Like CM in MPICH-V1, the mailbox script is a multi-threaded application that handles a set of TCP connections using a pool of threads In MPICH-V1, each MPI application is associated with a dispatcher process. Thus the dispatcher in MPICH-V1 architecture associated with an application, can also serve as the dispatcher in the our scheduler framework and it resides in the same master

node of the location which all selected resources to execute the processes of the application are directly located under its control. Thus, the MPICH-V1 dispatcher is modified for this purpose. Also, the mailboxes in our approach can serve as CMs in the MPICH-V1.

For a given application, MPICH-V1 dispatcher resides on the same master node along with our scheduler (in a site, zone, or super grid scheduler) and they communicate through a FIFO pipe. Our scheduler writes its migration commands into a FIFO pipe. The migration command consists of the rank of the process to be moved, the *ip* address of the destination node. The dispatcher checks for any data to read from the pipe.

The scheduler that all the selected resources are directly located under its control queries the information system periodically to get the updated information for those resources. If a failure is detected, the scheduler reschedules only the affected processes and re-executes them on the new resource from its last saved checkpoint and then the process retrieves the lost messages from its associated mailbox.

4. Experimentation and results:

To the best of our knowledge, a generally accepted set of benchmarks relative to fault tolerant scheduling of parallel applications with inter-process on heterogonous grid does not exist. However, some researchers have used population-based optimization algorithms to solve mapping problems. In small size grid, they indeed exhibits remarkable performance in optimizing in terms of final accuracy and robustness, however, due to the time- consuming nature of them, they are not the most suitable for online mapping in large-scale grid. Here, we will compare our approach in some cases with the work proposed by [14] since it is the closest study to our approach. In this work, an adaptive multisite mapping for computationally intensive grid applications based on multi-objective differential evolution algorithm.

In this work, any mapping solution can be represented by a vector μ of N integers ranging in the interval $[1,R]$, where R the total number of resources in grid and N is the number of processes in the application. Furthermore, there are two fitness functions, one accounting for the time of use of resources and the other for their reliability. Denoting with $T_{exe(i,j)}$ and $T_{comm(i,j)}$ respectively the computation and the communication times requested to execute the process i on the node j it is assigned to, the total time needed to execute i on j is: $T_{tot(i,j)} = T_{exe(i,j)} + T_{comm(i,j)}$, This is evaluated on the basis of available computation power and available bandwidth in the node *j*, which is the time spent by node *j* in executing computation and communication of the process *i* assigned to it by the proposed solution. So, the first fitness function is $\Phi_1(\mu) = \max_{j\in[1,N]} T_{tot(i,j)}$.

So, the goal of the proposed evolutionary algorithm is to search for the smallest fitness value among these maxima optimizing, in terms of time, the grid resource use. For

reliability, the second fitness function is given by the reliability of the proposed schedule (mapping). It is evaluated as: $\Phi_2(\mu) = \prod_{i=1}^{N} \pi_j \cdot \lambda_j$, Where π_j is the reliability of the computing resource onto which the ith process is mapped and λ_j is the reliability of link of site this node belong to. In our simulation, we have calculated the reliability of a given resource as a function of its probability of failure.

Finally, it should be pointed out that the first fitness function should be minimized, while the second should be maximized.

For comparison, we will keep the same parameters values for the proposed multi-objective differential evolution algorithm used by the authors, except for number of generation g which we should adjust so as to make the comparison with our approach in some possible cases. For brevity, we will refer to this work in the rest of the paper as multi-objective *DE* approach.

4.1 Simulation setup:

We have developed a simulator in *C++* to evaluate the performance of the proposed scheduling algorithm. In the simulation, our grid consists of ten sites denoted by $s_1, s_2, \ldots, s_i \ldots s_{10}$, each with R_i computing resources with heterogeneous raw CPU power (ranging from 600 to 1100 MIPS) connected by switched LAN, which in turn , are grouped into six zones denoted by $z_1, z_2, \ldots, z_l \ldots z_{10}$ as follow:
Sites s_1, s_2 are located in zone z_1, sites s_3, s_4, s_5 in z_2, site s_6 in zone z_3, site s_7 in zone z_4, and sites s_8, s_9, s_{10} in zone z_5. The intra-site maximum bandwidths of the sites s_1 and s_4 are 200 Mbps, where all of the other sites has intra-site maximum bandwidth of 100 Mbps. The intra-zone maximum bandwidths of the five zones, starting from zone z_1, are 30, 20,20,20,30 respectively, while the inter-zone maximum bandwidth between all zone is 3Mbps.

We simulate the Globus Meta Directory Service as a means to get grid resource loads and availability periodically at a predefined interval time. So, we generate these values with randomly percentage which, for each resource, keep constant for some interval before changing. The load percentage of the CPU of a resource (or the network link) indicates the availability degree of that resource and it ranges from 0 to 1.0 where a value of 1.0 indicates an unloaded resource and a value of 0 indicates the presence of a failure. Thus, the load percentage of 0.75 for a processor in our simulation setup represents one-fourth loaded of its raw CPU speed (in MIPS). This case is also the same for the availability of the links between resources.

Different simulations were carried out on two synthetic compute-intensive parallel applications with inter-process communication which are consists of 12 processes and 24 processes respectively, and denoted by A , B.

4.2 Comparison of mapping effectiveness and scalability of the two approaches

The first simulation was carried out on the two applications (A , B) to compare the mapping effectiveness and scalability of our hierarchical approach and the multi-objective DE approach for different grid size without considering the fault tolerance issue (we assume that the reliability of all resource is 100%), and then compute the performance metrics which are the total prediction completion time and the corresponding time of scheduling to get this schedule. For different approaches, the test for each size is made ten times for different CPU loads and the average has been taken for the comparison.

For each test in case of the multi-objective DE approach, we have continued increasing the number of generation *g* till the algorithm reach saturation and couldn't get any improvement in mapping solution (i.e. decreasing in the predicted total completion time).

The figures 4, 6 show the effect of increasing of number of resources on the predicted total completion time for the two applications respectively in the two approaches, where the figures 5,7 show the scheduling time needed to get corresponding mapping solution for each. The *Y-axis's* in the figures 4, 6 represent the predicted total completion time and the scheduling time to get the corresponding schedules respectively, while the *x-axis's* represent the number of resources in grid. (for the sake of justice comparison, we use the same number of resources in each site for every test i.e. 100 means, 10 computing resources in each site and 200 means, 20 computing resource in each site, and so on). While the *Y-axis's* in the figures 5, 7 represent the scheduling time needed by the scheduling algorithm to get the corresponding mapping solutions (schedules).

It is quite clear that the effect of increasing the number of computing resources (i.e. increasing the search space) on the predicted total completion time and also on the scheduling time of the both approaches, but it is observed from the figures that the differential evolution gives better schedules and reasonable scheduling time when the number of resources is small and becomes less effective as the number of resources increases although the multi-objective *DE* approach has global view since it is based centralized scheme. In despite of its hierarchical nature, our approach gives better schedule as the number of resources increases and also gives less scheduling time for all cases. Although there is a lack of global view in our scheduling approach, it gives better schedule because this class of applications with inter-process communication often has a tendency to execute within single site or zone to avoid the inter-site low bandwidth. Also, the improvement in scheduling time in our approach comes from the hierarchical scheme followed.

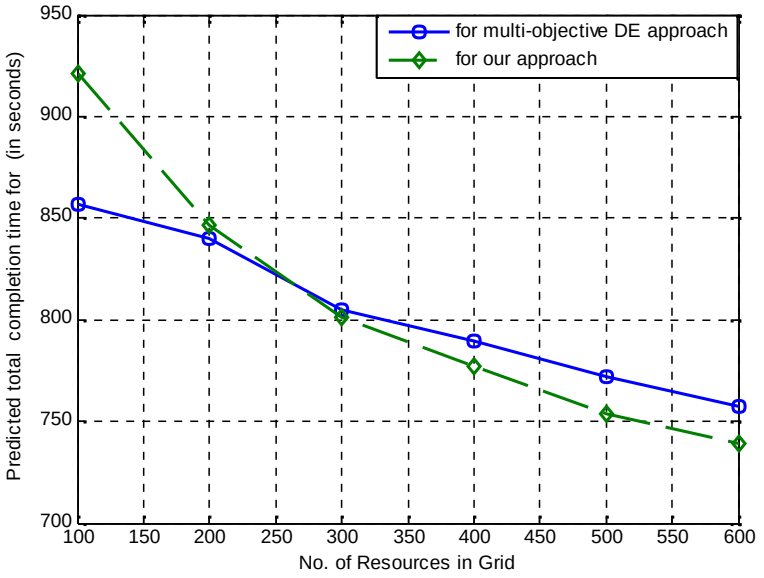


Figure (4): *The effect of increasing the number of resources on actual total completion. time for application A (12-process).*

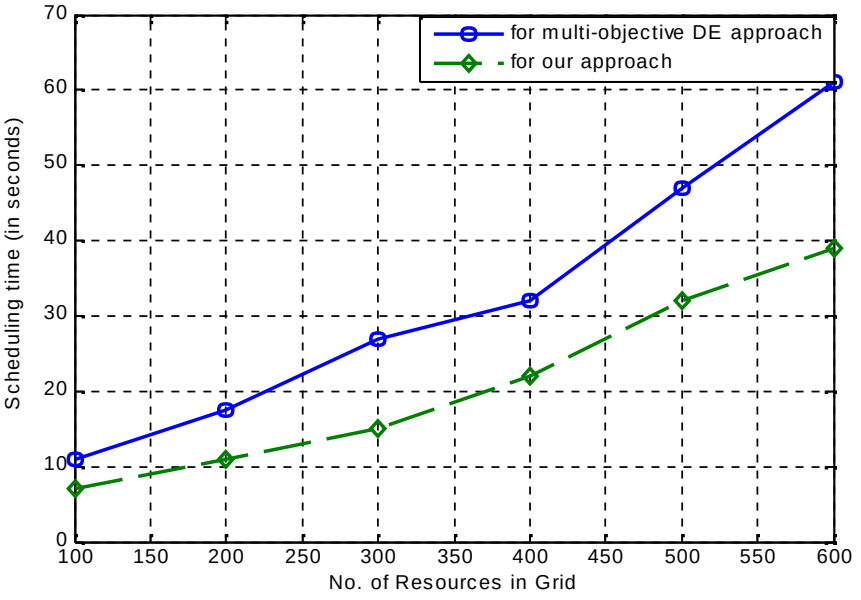


Figure (5): *The effect of increasing the number of resources on Scheduling time for application A (12-process)*

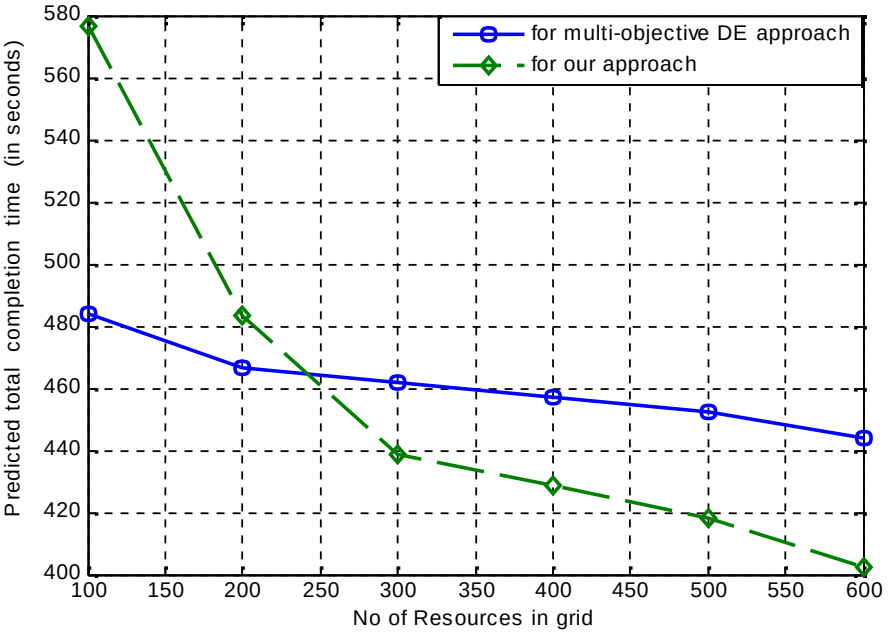


Figure (6): *The effect of increasing the number of resources on actual total completion time for application B (24-process)*

4.3 Comparison the two approach with considering the fault tolerance:

This experiment was to show that the post-active fault tolerance technique we have integrated in our approach by using MPICH-V1 to process checkpointing and migration only the affected process from failed node to another yields more benefit for long running applications compared to the pro-active technique used by the multi-objective DE approach in highly fault-prone and volatile environment even if the failures occurrence can be predictable. The experiment was carried out on the two applications A , B; but in this paper, we have reported the results only of the application A .
In this experiment, we simulate grid of fixed size -300 computing resources as 30 resources in each site (we took this size because the two approaches gave almost the same predicted total completion time in the first experiment). Also, we have generated random errors with different probabilities to simulate computing resource failures in the

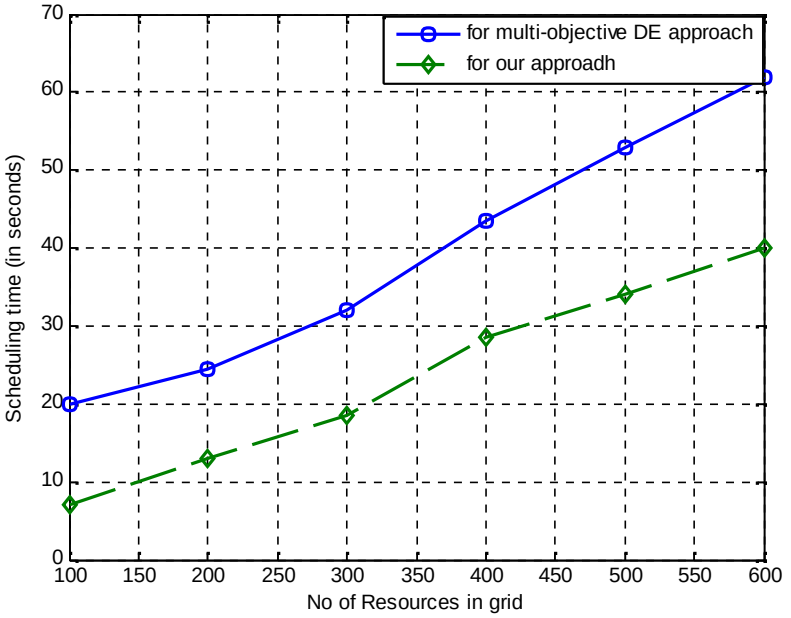


Figure (7): *The effect of increasing the number of resources on Scheduling time for application B (24-process)*

trace file of Meta Directory Service and compute the reliability of resource accordingly, and then we have compared the actual total completion time for the two approaches. For the sake of simplicity, for each test we have generated the same failure probability for all computing resources with even IDs, while all resources with odd IDs have the same different failure probability.

For our approach, each test was carried for three different simulated checkpoint overhead of 10, 15, 20 seconds/checkpoint, while the checkpoint interval have fixed at 130s period in all simulations, while for multi-objective DE approach , we have used the same values of the parameters needed by the multi-objective differential evolution mapping algorithm as suggested by the authors.

For each failure probability case, each test is made ten times for different CPU loads and the average has been taken for the comparison. Figure 8 shows the effect of increasing the failure probabilities of resources on the actual completion time for application A (12 processes), in which Y-axis represents the actual total completion time, while the x-axis represents resources failure probabilities.

The figure 8 shows that our approaches gives better performance for almost cases compared to the multi-objective DE approach except for few cases when failure probability is small due to the checkpoint overhead.

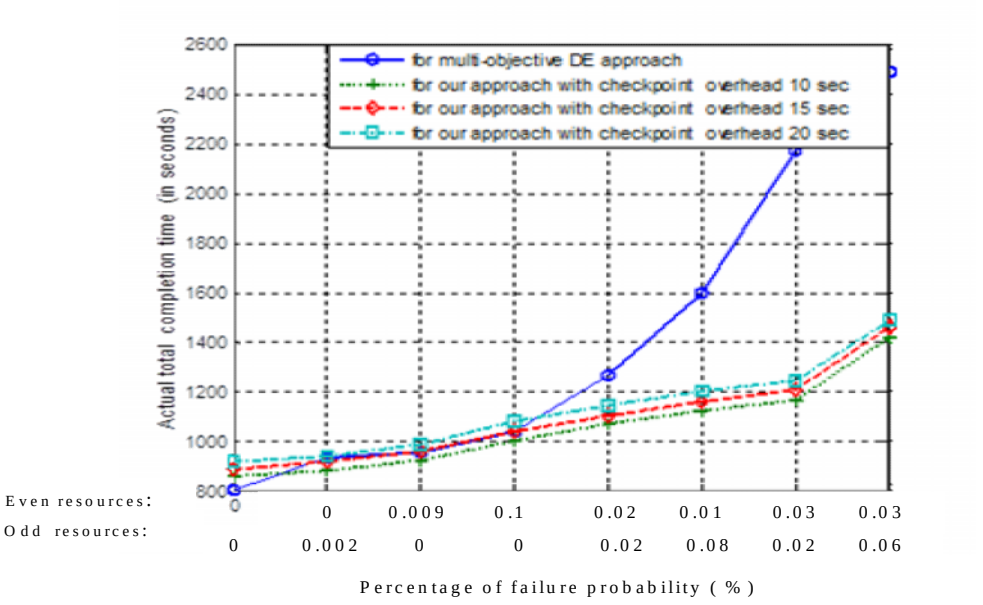


Figure (8): *The effect of failure probability on actual total completion time for application A (12-process).*

5. Conclusion

In this paper, a new fault-tolerant and scalable scheduling scheme for parallel applications with heterogeneous inter-process communication on high dynamic and heterogeneous grid is presented. Applications' processes indirectly communicate through new proposed mailbox-based communication model at a shared node. The proposed mail-box model is exploited by our scheduling to simplify the mapping process of application's processes to the best resource in salable manner. MPICH-V1 is slightly modified and integrated into our scheduling system which also exploits the mail-box pool as Channel Memory (CM). The performance of the proposed strategy is evaluated using a simulator developed in C++. The experimental result demonstrate that the proposed scheduling effectively schedule parallel applications with inter-process communication in fault tolerant and scalable way in spite of highly dynamic nature and largely size of grid.

It is also observed that, for highly volatile grid, the performance of the systems that integrate post-active fault tolerance mechanism into scheduling often outperform the performance of systems that use pro-active fault tolerance mechanism because of the frequently and unpredictable failures occurrence. The future work will focus on improving our scheduling hierarchical tree by relieving the assumption stating that the master nodes are dedicated nodes and don't fail by making the tree dynamically re-configurable and fault tolerant.

References:

[1] M.A. Al-fawair, "A Framework for Evolving Grid Computing Systems," Ph.D. dissertation, Faculty of Technology. De Montfort University, United Kingdom," 2009.

[2] M. Nandagopal, "FAULT TOLERANT SCHEDULING STRATEGY FOR COMPUTATIONAL GRID ENVIRONMENT," *International Journal of Engineering Science*, vol. 2, 2010, pp. 4361-4372.

[3] S. Nasir, M. Shah, A. Kamil, B. Mahmood, and A. Oxley, "Hybrid Resource Allocation Method for Grid Computing," *Second International Conference on Computer Research and Development*, 2010, pp. 0-5.

[4] H. H.Mohamed, "The Design and Implementation of the KOALA Grid Resource Management System," 2007.

[5] A.Tchernykh, U. Schwiegelshohn, R. Yahyapour, and N. Kuzjurin, "On-line hierarchical job scheduling on grids with admissible allocation," *Journal of Scheduling*, vol. 13, Mar. 2010, pp. 545-552.

[6] S. a Jarvis, D.P. Spooner, and G.R. Nudd, "Dynamic Scheduling of Parallel Jobs with QoS Demands in Multiclusters and Grids," *Proceedings of the Fifth IEEE/ACM International Workshop on Grid Computing (GRID'04)*, 2004, pp. 402-409.

[7] L. CHEN, "Process migration and runtime scheduling for parallel tasks in computational grids." Ph.D. dissertation.University of Hong Kong," 2007.

[8] G.S. Choi, J.-ha Kim, D. Ersoz, and C.R. Das, "Coscheduling in Clusters : Is It a Viable Alternative ?," *Supercomputing, 2004. Proceedings of the ACM/IEEE SC2004 Conference*, 2004, p. 16.

[9] P.G. Sobalvarro, S. Pakin, W.E. Weihl, and A.A. Chien, "Dynamic Coscheduling on Workstation Clusters," *Systems Research*, 1998, p. 231--256.

[10] E. Frachtenberg, D.G. Feitelson, F. Petrini, and J. Fernandez, "Adaptive Parallel Job Scheduling with Flexible CoScheduling," *Transactions on Parallel and Distributed Systems, IEEE*, 2005, pp. 1066 - 1077.

[11] S. Nagar, A. Banerjee, A. Sivasubramaniam, and C.R. Das, "A closer look at coscheduling approaches for a network of workstations," *Proceedings of the eleventh annual ACM symposium on Parallel algorithms and architectures – '99*, New York, USA: ACM Press, 1999, pp. 96-105.

[12] Z.L. Qian Zhang, "Design of Grid Resource Management System Based on Information Service," *JOURNAL OF COMPUTERS*, vol. 5, 2010,pp. 687-694.

[13] Y. Zhang, A. Mandal, C. Koelbel, K. Cooper, and C. Hill,"Combined Fault Tolerance and Scheduling Techniques for Workflow Applications on Computational Grids," *9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, 2009, pp. 244-251.

[14] I. De Falco, U. Scafuri, and E. Tarantino, "An adaptive multisite mapping for computationally intensive grid applications," *Future Generation Computer Systems*, vol. 26, Jun. 2010, pp. 857-867.

[15] S. Augustin, P. Wieder, and W. Ziegler, "A Meta-Scheduling Service for Co-allocating Arbitrary Types of Resources," *Grid Resource Management Workshop 2005, Parallel Processing and Applied Mathematics, 6th International Conference*, 2005, pp. 782-791.

[16] N.A. Mehdi, A. Mamat, H. Ibrahim, and S.A.P. K, "Multiphase Scalable Grid Scheduler Based on Multi-QoS Using Min-Min Heuristic,"*(IJACSA) International Journal of Advanced Computer Science and Applications*, vol. 1, 2010, pp. 10-14.

[17] B. Nazir and T. Khan, "Fault Tolerant Job Scheduling in Computational Grid," *International Conference on Emerging Technologies, IEEE*, 2006, pp.708-713.

[18] M.J. Litzkow, M. Livny, and M.W. Mutka, "Condor-a hunter of idle workstations," *Proceedings of The 8th International Conference on Distributed Computing Systems.* Soc. Press, 1988, pp. 104-111.

[19] H.C. Krijn Raadt, Yang Yang, "Practical Divisible Load Scheduling on Grid Platforms with APST-DV," *In Proc. of the 19th International Parallel and Distributed Processing Symposium – IPDPS'05*, 2005, p. 29b.

[20] S.S.V. H. A. Sanjay, "A strategy for scheduling tightly coupled parallel applications on clusters," *Concurrency and Computation:Practice and Experience*, vol. 21, 2009, p. 2491–2517.

[21] M. Pasquali, R. Baraglia, G. Capannini, L. Ricci, and D. Laforenza, "A multi-level scheduler for batch jobs on grids," *The Journal of Supercomputing*, vol. 57, Feb. 2011, pp. 81-98.

[22] C.-T. Yang, K.-Y. Chou, and K.-C. Lai," Design and implementation of an adaptive job allocation strategy for heterogeneous multi-cluster computing systems," *Concurrency and Computation: Practice and Experience*, vol. 23, 2011, p. 1701–1722.

[23] S.K. Dimitriadou and H.D. Karatza, "Multi-Site Allocation Policies on a Grid and Local Level," *Electronic Notes in Theoretical Computer Science*, Elsevier B.V., 2010, pp. 163-179.

[24] V. Kravtsov, P. Bar, D. Carmeli, A. Schuster, and M. Swain, "A scheduling framework for large-scale, parallel, and topology-aware applications," *Journal of Parallel and Distributed Computing*, vol. 70, Sep. 2010, pp. 983-992.

[25] G.A.-R.G.E.M. El-Sayed, "A High Performance Dynamic Meta-Scheduler with Migration Support for Grid Applications," 2007.

[26] F. Berman, A. Chien, K. Cooper, J. Dongarra, I. Foster, D. Gannon, L. Johnsson, K. Kennedy, C. Kesselman, J. Mellor-crummey, D. Reed, L. Torczon, and R. Wolski, "The GrADS Project: Software Support for High-Level Grid Application Development," *International Journal of High Performance Computing Applications*, vol. 15, 2001, p. 327–344.

[27] H. Dail, "A decoupled scheduling approach for Grid application development environments," *Journal of Parallel and Distributed Computing*, vol. 63, May. 2003, pp. 505-524.

[28] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, a Su, and D. Zagorodnov, "Adaptive computing on the grid using AppLeS," *IEEE Transactions on Parallel and Distributed Systems*, vol. 14, Apr. 2003, pp. 369-382.

[29] F.B.A.R. Wolski, "The Apples Project: A Status Report," *Proceedings of the Eight NEC Research Symposium, Germany, May*, 1997, p. 19.

[30] Henri Casanova and Jack Dongarra, "NetSolve: A network Server for Solving Computational Problems," *The International Journal of Supercomputer Applications and High Performance Computing*, vol. 11, 2000, p. 212-223.

[31] J.H. Abawajy, "Fault-Tolerant Dynamic Job Scheduling Policy," *ICA3PP*, 2005, pp. 165-173.

[32] M. Nandagopal, "Hierarchical Status Information Exchange Scheduling and Load Balancing For Computational Grid Environments," *Journal of Computer Science*, vol. 10, 2010, pp. 177-185.

[33] J. Cao, S.A. Jarvis, S. Saini, G.R. Nudd, and S. Augustin, "GridFlow : Workflow Management for Grid Computing," *Cluster Computing and the Grid, 2003. Proceedings. CCGrid*, 2003, pp. 198 - 205.

[34] M.A. E. Fern´ Andez, E. Heymann, "Scheduling for Interactive and Parallel Applications on Grids," 2008.

[35] S. Prabhakar, "Zone Based Scheduling : A Framework for Scalable Scheduling of SPMD parallel programs on the Grid," 2003.

[36] F. Berman, R. Wolski, S. Figueira, J. Schopf, and G. Shao, "Application-Level Scheduling on Distributed Heterogeneous Networks(Technical Paper)," *Proceedings of Supercomputing ACM/IEEE*, 1996, pp. 1-29.

[37] H. Dail, R. Wolski, and A. Grimshaw, "Application-Aware Scheduling of a Meg-netohydrodynamics Application in the Legion Metasystem," Heterogeneous Computing Workshop. *(HCW 2000) Proceedings. 9th, 2000*, pp. 216 - 228.

[38] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, a Su, and D. Zagorodnov,"Adaptive computing on the grid using AppLeS,"*IEEE Transactions on Parallel and Distributed Systems*, vol. 14, Apr. 2003, pp. 369-382.

[39] "GT4.1.3 information services (MDS) http://www.globus.org/toolkit/docs/development/4.1.3/."

[40] R. Wolski, T. Spring, and J. Hayes,"The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing," *Journal of Future Generation Computing Systems*, vol. 15, 1999, p. 757--768.

[41] P. Lemarinier, "MPICH-V2 : a Fault Tolerant MPI for Volatile Nodes based on Pessimistic Sender Based Message Logging," *Proceedings of the ACM/IEEE SC2003 Conference (SC'03)*, 2003.

[42] T. Herault, P. Lemarinier, F. Cappello, and I. Lri, "MPICH-V Project : a Multiprotocol Automatic Fault Tolerant MPI," 2001, pp. 1-12.

[43] M.V. Reddy and S. Chaudhary, "Scheduling in Grid: Rescheduling MPI applications using a fault-tolerant MPI implementation," *2nd International Conference on Communication Systems Software and Middleware, IEEE*, 2007, pp. 1-8.

[44] P. Lemarinier and D.P. Sud, "MPICH-V2 : a Fault Tolerant MPI for Volatile Nodes based on Pessimistic Sender Based Message Logging," *Proceedings of the ACM/IEEE SC2003 Conference (SC'03)*, 2003.

[45] G. Bosilca, A. Bouteiller, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, and A. Selikhov, "MPICH-V : Toward a Scalable Fault Tolerant MPI for Volatile Nodes," *Proceedings of the IEEE/ACM SC2002 Conference (SC'02)*, 2002.