J Supercomput (2010) 51: 97–114
DOI 10.1007/s11227-009-0276-7

An adaptive task-level fault-tolerant approach to Grid

Yongwei Wu · Yulai Yuan · Guangwen Yang · Weimin Zheng

Published online: 14 March 2009

Abstract A strong failure recovery mechanism handling diverse failures in heterogeneous and dynamic Grid is so important to ensure the complete execution of long-running applications. Although there have been various efforts made to address this issue, existing solutions either focus on employing only one single fault-tolerant technique without considering the diversity of failures, or propose some frameworks which cannot deal with various kinds of failures adaptively in Grid. In this paper, an adaptive task-level, fault-tolerant approach to Grid is proposed. This approach aims at handling quite a complete set of failures arising in Grid environment by integrating basic fault-tolerant approaches. Moreover, this paper puts forward that resource consumption (not received enough attention) is also an important evaluation metric for any fault-tolerant approach. The corresponding evaluation models based on mean execution time and resource consumption are constructed to evaluate any fault-tolerant approach. Based on the models, we also demonstrate the effectiveness of our approach and illustrate the performance gains achieved via simulations. The experiments based on a real Grid have been made and the results show that our approach can achieve better performance and consume less resource.

Keywords Grid · Task-level · Fault-tolerant · Resource consumption · Mean execution time

Y. Wu (✉) · Y. Yuan · G. Yang · W. Zheng
Department of Computer Science and Technology, Tsinghua National Laboratory for Information Science and Technology, Tsinghua University, Beijing, 100084, People's Republic of China
e-mail: wuyw@tsinghua.edu.cn

Y. Yuan
e-mail: yuan-yl05@mails.tsinghua.edu.cn

G. Yang
e-mail: ygw@tsinghua.edu.cn

W. Zheng
e-mail: zwm-dcs@tsinghua.edu.cn

1 Introduction

Grid connects PCs, workstations, clusters and many other high performance computers through networks to provide huge computational and storage capabilities transparently [11–13]. It makes constructing large-scale applications possible in the fields of high-energy physics, gravitational-wave physics, bioinformatics, etc. However, developing, deploying, and running programs on Grid are of significant challenges due to diverse failures encountered during execution.

Failures due to the inherently unreliable nature of Grid environment include task-level failure and Grid-level failure. By task-level failure we mean that a task ceases to run so that the fact that the task has failed is to be recognized by, e.g., the coordinator task of the Grid application. Task-level failure denotes a high-level aggregate abstraction of failure type that encapsulates an large number of anonymous low-level failures, e.g., node crash, OS failures, host shutdown for regular maintenance. This type of failure is referred to as fail-stop crash failure [16]. We are aware of other mode of failures, Byzantine failures, in which the task operates in a malicious manner. In this paper, we do not focus on the Byzantine failures. Grid-level failures appear on Grid system itself such as Grid middleware or Grid workflow engine crashes. Since tasks are the main visible entity that needs to be paid attention to when we are developing and running Grid applications, task-level failures are our main concern.

Task-level fault-tolerant techniques have been widely studied in parallel and distributed systems. Retry, alternate resource, checkpoint/restart and replication are the four basic approaches [1, 9]. At present, either a single approach of the four is used, or a combination of two or three is applied [1–3, 5, 7, 8, 14, 17, 18, 20, 22]. However, all these methods are not able to handle the dynamic environment of Grid. Once a single approach or a combination is selected, they cannot be changed during run time. Thus the capability of handling faults in a given system is limited.

In this paper, we propose an adaptive task-level, fault-tolerant approach to Grid. It is based on the four basic approaches, and it can dynamically and automatically decide which one is used by analyzing the current state of the running task. An evaluation model (for mean execution time) is constructed and used to evaluate fault-tolerant approaches. Moreover, resource consumption is newly introduced as the other important metric for evaluation of fault-tolerant approaches. Corresponding evaluation models for resource consumption are also constructed.

Simulations have been performed to compare our approach with the basic four approaches and different combinations of them. Also, a series of experiments are carried out on a real Bioinformatics Grid (BioGrid) [10]. Simulations and experiments both show that the mean execution time of a program using our approach is very close to the time the program spends in a failure-free environment. Moreover, our approach consumes less resource.

The rest of the paper is arranged as follows. Section 2 introduces the related work. Our approach is described in detail in Sect. 3 along with the analysis of limitations of existing approaches. Evaluation model of the mean execution time of our approach is presented in Sect. 4. Resource consumption of retry, alternate resource, checkpoint/restart, replication and our approach are also analyzed in this section. Section 5 demonstrates the simulation results. Experiments based on a real Bioinformatics Grid (BioGrid) are described in Sect. 6. Section 7 draws conclusions.

2 Related work

A lot of research has been done on fault-tolerant mechanisms in distributed, parallel, and Grid systems. In this section, we present a brief overview of the related work in the area of task-level fault-tolerant techniques for Grid systems.

Some systems use a single one of the basic four approaches. Condor-G [8] adopts retry on the same machine as its fault-tolerant mechanism. DOME [1] is implemented using checkpoint/restart. However, retry and checkpoint/restart alone are not powerful enough for distributed environments. For example, if the node a task running on crashes, this task cannot complete properly. Finding another node and restarting the task on it seems able to solve such problems. Netsolve [3], GridFlow [2], and Gridbus workflow [22] are implemented using alternate resource as their fault-tolerant mechanism. Actually it is not easy to find a proper node. For example, in a failure-intensive condition, it is very possible for a system to find another crashed node so that it will keep looking for proper nodes infinitely. This could be a big overhead.

Since methods with single approach are not sufficient for Grid environments, some systems adopt a combination of two or three of the four basic approaches. Askalon [7] and Taverna [14] use retry and alternate resource; DAGMan [17], Pegasus [5] and Triana [18] apply migration-based retry and checkpoint approaches; Karajan [20] adopts retry, alternate resource and checkpoint. However, these approaches (single and combination) do not take into account the dynamic property of Grid. Once the approach is selected, it will be applied to the whole process and cannot be changed. This is a big limitation for a given system.

S. Hwang [9] proposes a flexible fault handling framework considering heterogeneous and dynamic Grid environment. But it heavily relies on the users' configuration. Users are required to specify the fault-tolerant mechanism in advance for each task. Thus users must know the exact status of tasks and environments. This is a very high expectation for ordinary Grid users.

3 Adaptive fault-tolerant approach

In whole, retry, alternate resource, checkpoint/restart and replication are the four basic fault-tolerant approaches for distributed environment. Their limitations could be summarized as follows.

- **Retry:** keeping retrying on the same node cannot solve some problems such as node crashing, network disconnection, etc.
- **Alternate resource:** if the environment is prone to failure, the system using this approach needs to find a new node and deploy tasks on it very frequently. This is a big overhead. Even worse, the system may fall into the situation of keeping looking for a proper node infinitely.
- **Checkpoint/restart:** checkpoint can make a task, especially long running task, work correctly. However, checkpoint alone cannot be robust enough to handle all kinds of failures such as node crashing. It should work together with other approaches, such as migration (alternate resource), in order to achieve better performance.

Table 1 The adaptive fault-tolerant approach

1. All tasks are assigned an initial *failure_rank* of 3, meanwhile the checkpoint subsystem is initialized and started (the time interval of checkpoint can be configured by users who submit tasks)
2. Whenever a failure occurs, the system decides which action to take based on the *failure_rank* of the task.
   2a. if *failure_rank* = 3, 'retry' will be used
   - Retry the task from the latest checkpoint
   - *failure_rank* − −

   2b. if *failure_rank* = 2, 'alternate' resource' will be used
   - Find an alternate node, deploy the task and transfer the checkpoint file
   - After switching to the new node, restart the task from the latest checkpoint
   - *failure_rank* − −

   2c. if *failure_rank* = 1, 'replication' will be used
   - Find N nodes for deploying task replicas on and transfer checkpoint file simultaneously (N can be configured)
   - Restart replicas simultaneously on N new deployed nodes from the latest checkpoint

- **Replication:** making N replicas to ensure a task to complete seems to be the safest fault-tolerant approach. However, it consumes too many resources. Just imagine that when a task has been deployed on N nodes, and each node works correctly, as a result it will consume N times of resources than failure-free execution.

An adaptive fault-tolerant approach should be able to handle diverse failures dynamically and automatically. Based on the characteristics of Grid and the limitations of the four basic approaches discussed above, we have identified three fundamental requirements for Grid:

- **Handling diverse failures:** Grid fault-tolerant approach should solve a complete set of failures which may be encountered by one application.
- **Low overhead:** using fault-tolerant approaches may bring in extra overhead, including failure detection and failure handling. The overhead of a Grid fault-tolerant approach should be small as possible.
- **Low resource consumption:** a Grid fault-tolerant approach should consume extra resource as little as possible.

Table 1 describes our fault-tolerant approach. It is a combination of the four basic ones. During the whole process of a task, checkpoint is taken as a daemon process. If first failure occurs, our approach retries the task from the latest checkpoint on the same node. The failure can be considered to be a transient one, so retry is the simplest and cheapest way to handle it. If second failure occurs, we have the reason to consider the node on which the task running is not stable. Thus, alternate resource is the second step. If alternate resource still cannot make the task complete, it is believed that the environment is prone to failures. Finally replication is applied, and N replicas of the same task are running simultaneously.

4 Evaluation model

Mean execution time of a task is the most fundamental and intuitional metric for evaluating any fault-tolerant approach. We consider resource consumption as another important metric at the same time. In this section, evaluation models for mean execution time and resource consumption are constructed. Before this, some important parameters and assumptions are introduced.

4.1 Parameters and assumptions

Important parameters for analysis are described as follows:

Failure-free execution time of a task (F). This is the execution time a task needs in the absence of failures. F is a constant.

Mean Time To Failure (MTTF). *TTF* (Time To Failure) is a random variable representing the time between adjacent arrivals of failures. *MTTF* is a mean time of failure intervals (*TTF*).

Failure rate (λ). Failure occurrence obeys Poisson distribution. This is commonly assumed in fault-tolerant research [6, 20]. $MTTF = 1/\lambda$ [15].

Average Checkpoint Overhead (C). C is an average amount of time required to create a checkpoint. C is a constant.

Recovery Time (R). This is the time that the system takes to restore the task from the latest checkpoint. This parameter is a constant.

Number of Replicas (N). This is the replica number of a task executed simultaneously if replication is adopted.

Find and deploy a task on a node (T_f). This is the time for finding and deploying the failed task on another node. T_f is a constant.

Uninterrupted task execution time between checkpoints (α). This is the time interval between two adjacent checkpoints in the failure-free runs. Thus, if K checkpoints are created during F, $\alpha = F/K$.

λ. The instants of the occurrences of failures form a homogeneous Poisson process of parameter λ. Thus, the time interval between two adjacent failures is governed by Exponential distribution of parameter λ [20]. λ is a constant.

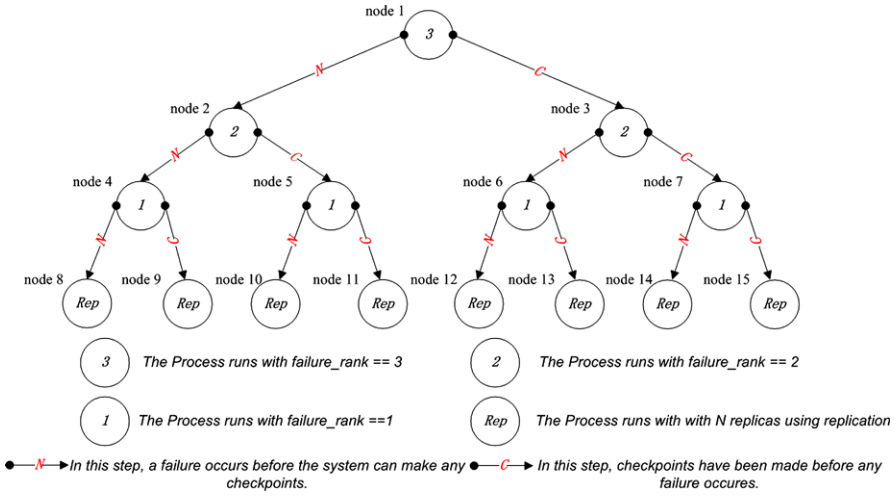


Fig. 1 State transition diagram of our approach

4.2 Mean execution time model

An evaluation model for calculating mean execution time of our approach is constructed in this part. Figure 1 is the state transition diagram tree of our new approach.

Figure 1 is the state transition diagram of our approach. The number k in the circle means the task runs with $failure\_rank = k$ on the node (for example, a task runs on the root of the tree with $failure\_rank = 3$). The transition/arrow presents that a failure occurs and the task transits to the next state. N on the transition arrow denotes that a failure occurs and no checkpoint has been made, whereas C denotes that some checkpoints have been made before any failures. "node i" means this is the ith node in the tree (nodes are numbered by level order of the tree).

A task may finish at any state. It can run without any failures (the root node) or arrive at any leaf node. There are 15 possible endings for a task (15 nodes in Fig. 1). T_i denotes execution time of a task which terminates at the ith node of the tree, and P_i is the probability of such case. T is the execution time of a task. Obviously, the mean execution time of the new approach is:

$$E[T] = \sum_{i=1}^{15} T_i \cdot P_i. \tag{1}$$

Without the loss of generality, we take T_2 as an example. If a task completes at state 2, it first runs with $failure\_rank = 3$ on a node, during which no checkpoint has been made. Then a fault occurs and the task is retried. Finally, it finishes at state 2. Thus, $T_2 = t_3 + F'$, $P_2 = e^{-\lambda \cdot F'} \cdot (1 - e^{-\lambda \cdot b})$.

A t_3 means the time a task runs with $failure\_rank = 3$. We define b the time between ends of two adjacent checkpoints, $b = \alpha + C$. F' denotes the actual time a task needs with overhead of checkpoint, $F' = F + C \cdot F/\alpha$. It is a random variable governed by Truncated Exponential Distribution of parameter (λ, b). The pdf

(Probability Distribution Function) of t_3 is $f_{t3}(x|\lambda, b) = \lambda \cdot e^{-\lambda \cdot x}(1 - e^{-\lambda \cdot b})$. And, $E[t_3] = 1/\lambda - b/(e^{\lambda \cdot b} - 1)$.

$T_1 \sim T_{15}$ and $P_1 \sim P_{15}$ can be calculated using the same method. Thus, the mean execution time of a task using the new fault-tolerant approach is:

$$E[T] = F' + P_{T_\mathrm{f}} \cdot T_\mathrm{f} + P_\lambda \cdot \left(\frac{1}{\lambda} - \frac{b}{e^{\lambda \cdot b} - 1}\right) + P_R \cdot R \tag{2}$$

where $P_{T_\mathrm{f}} = (2 - p_1) \cdot (1 - p_1)^2$, $P_\lambda = (1 - p_1) + (1 - p_1)^2 + (1 - p_1)^3$, $P_R = 3p_1p_2p_3 + p_1p_2 + 2p_1p_2^2 + 6p_2p_3^2 + 8p_2^2p_3 + 3p_3^3$, $p_1 = e^{-\lambda \cdot F'}$, $p_2 = e^{-\lambda \cdot b} - e^{-\lambda \cdot F'}$, $p_3 = 1 - e^{-\lambda \cdot b}$.

4.3 Resource consumption model

Although Grid provides a huge and powerful virtual PC to users, resources, especially computational resources are still limited.

Defining metrics for resource consumption of a fault-tolerant approach is not easy, because there are too many 'resources', including CPU, memory, I/O, etc. In this paper, only CPU consumption is taken into account; However, the approach computing CPU consumption can also be easily adapted to cover other types of resource consumptions.

CPU resource consumption is measured by the total CPU time used by a task. If a task runs on 5 nodes and t_i is the execution time of a task running on node i, the resource consumption of this task is $\sum_{i=1}^{5} t_i$.

- Resource consumption of retry, alternate resource, checkpoint/restart and replication

  As retry, alternate resource and checkpoint/restart do not need any replicated task, resource consumption of them can be measured by their mean execution time T_retry, $T_{CP/R}$, $T_\mathrm{alternate}$. T_retry, $T_{CP/R}$ are calculated by Duda [6]. Thus, the resource consumption R_retry, $R_{CK/R}$ of retry and checkpoint/restart are:

$$E[R_\mathrm{retry}] = E[T_\mathrm{retry}] = \frac{e^{\lambda \cdot F} - 1}{\lambda}$$

$$E[R_{CP/R}] = E[T_{CP/R}] = \frac{F}{\alpha} \cdot \left[C + \left(R + C + \frac{1}{\lambda}\right) \cdot (e^{\lambda \cdot \alpha} - 1)\right].$$

  Comparing alternate resource with retry, we find the only difference is that when a task fails, retry restarts the task on the same node with the recovery overhead R, while alternate resource needs to find and deploy the task on another proper node and restart it with the overhead T_f. Thus, the resource consumption $R_\mathrm{alternate}$ is:

$$E[R_\mathrm{alternate}] = E[T_\mathrm{alternate}] = \left(T_\mathrm{f} + \frac{1}{\lambda}\right) \cdot (e^{\lambda \cdot F} - 1).$$

Resource consumption of replication is the sum of all the running time of replicated tasks t_i. And t_i obeys the Truncated Exponential Distribution of parameter (λ, b). Thus, resource consumption of replication with N replicas is:

$$E[R_{\text{replication}}] = N \cdot \left(\frac{1}{\lambda} - \frac{F}{e^{\lambda \cdot F} - 1} \right).$$

- Resource consumption of our approach
  R_i denotes resource consumption of a task which terminates at the ith node of the tree in Fig. 1, and P_i is the probability of such case. R is the CPU resource consumption of this task. Obviously, the resource consumption is:

$$E[R] = \sum_{i=1}^{15} R_i \cdot P_i. \tag{3}$$

Obviously,

$$R_i = \begin{cases} T_i, & 1 \le i \le 7, \\ T_i - E[T_{\text{replication}}] + E[R_{\text{replication}}], & 8 \le i \le 16. \end{cases}$$

Thus, resource consumption of our approach is:

$$E[R] = P_{F'} \cdot F' + P_{T_{\text{f}}} \cdot T_{\text{f}} + P_\lambda \cdot \left(\frac{1}{\lambda} - \frac{b}{e^{\lambda \cdot b} - 1} \right) + P_R \cdot R + P_E \cdot E[R_{\text{replication}}] \tag{4}$$

where

$$\begin{aligned} P_{F'} &= 1 - (1 - p_1)^3 \\ P_{T_{\text{f}}} &= (1 - p_1)^2 \\ P_\lambda &= (1 - p_1) + (1 - p_1)^2 + (1 - p_1)^3 \\ P_R &= 3p_1p_2p_3 + p_1p_2 + 2p_1p_2^2 + N \cdot [(1 - p_1)^3 - p_3^3] \\ P_E &= (1 - p_1)^3 \\ p_1 &= e^{-\lambda \cdot F'},\ p_2 = e^{-\lambda \cdot b} - e^{-\lambda \cdot F'},\ p_3 = 1 - e^{-\lambda \cdot b}. \end{aligned}$$

5 Simulation

In this section, we use simulation to evaluate our approach compared with the basic four and combinations of the four basic fault-tolerant approaches. Mean execution time and resource consumption are the key metrics we concern.

5.1 Mean execution time

We make comparison between our approaches and the four basic fault-tolerant approaches and different combinations of the four. We simulate the mean execution

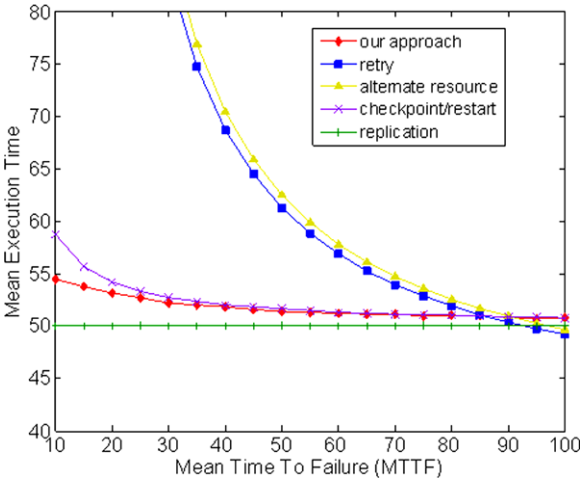


Fig. 2 Our approach vs. 4 basic approaches

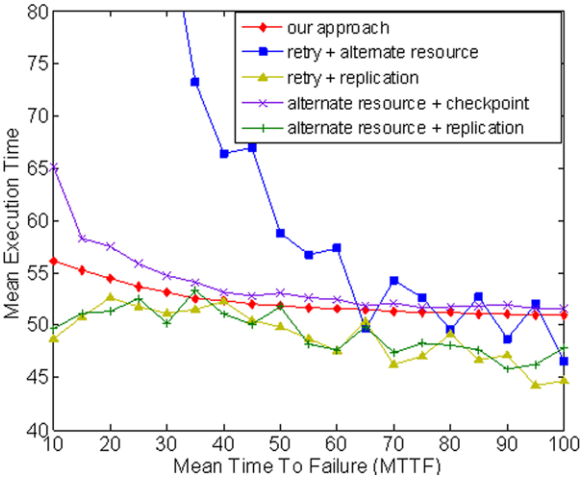


Fig. 3 Our approach vs. combinations of 2 of the four basic approaches

time (Y-axis) of the target task for various failure rates ($MTTF$, X-axis). For this simulation, we fixed parameter F to 40, K to 20, C to 0.5, R to 0.5 and T_f to 1. So this task needs $F' = 50$ to complete in a failure-free environment considering the over head of checkpoint.

Figure 2 shows the mean execution time of our approach compared with the four basic fault-tolerant approaches respectively. Our approach is compared with combinations of two approaches of the basic four in Fig. 3, and three in Fig. 4.

Figures 3 and 4 show that the mean execution time of a task using our approach is very close to the time the task running in a failure-free environment. And the mean execution time is almost constant with the change of $MTTF$. This is an ideal solution for Grid environment.

The mean execution time of a task without support of checkpoint or replication increases exponentially with failure rate ($\lambda = 1/MTTF$). Curves of checkpoint are also exponential to λ, but they do not grow as fast as retry and alternate resource.

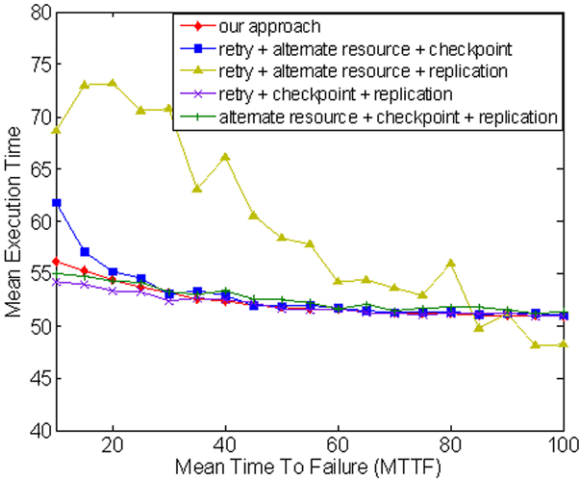


Fig. 4 Our approach vs. combinations of 3 of the four basic approaches

Checkpoint works well on various conditions. However, fault-tolerant approaches without support of replication cannot handle all kinds of failures as we stated in Sect. 2.

Curves of replication supported approaches and ours are almost identical. But our approach has some advantages of small resource consumption (to be discussed later) over replication.

5.2 Resource consumption

More parameters should be considered for simulating resource consumption of different fault-tolerant approaches. Besides *MTTF*, F is an important variable for resource consumption because different approaches react variably with the change of F. Replica number N for replication supported approaches can change the resource consumption dramatically.

Figures 5, 6, 7 depict resource consumption of different approaches with parameter F and *MTTF*. For replication supported approaches, replication number N is fixed to 5.

Figure 8 shows the trends of resource consumption with parameter N and *MTTF*. All these approaches are replication supported.

From these figures, we can conclude that checkpoint is a good approach to slow down the increase of resource consumption. Resource consumption of approaches only use retry or alternate grows exponentially. Replication supported approaches grows relatively slower than retry and alternate, because replication can ensure a task complete even the failure rate is high. But they perform badly when *MTTF* is high. This extra resource consumption is meaningless.

Figure 9 depicts resource consumption of our adaptive approach. Our approach is checkpoint and replication supported, so all above conclusions are true to our approach. However, our approach grows much slower than other replication supported approaches. In most conditions, the resource consumption is constant (near to F'). Unlike replication, our approach performs pretty well when *MTTF* is high.

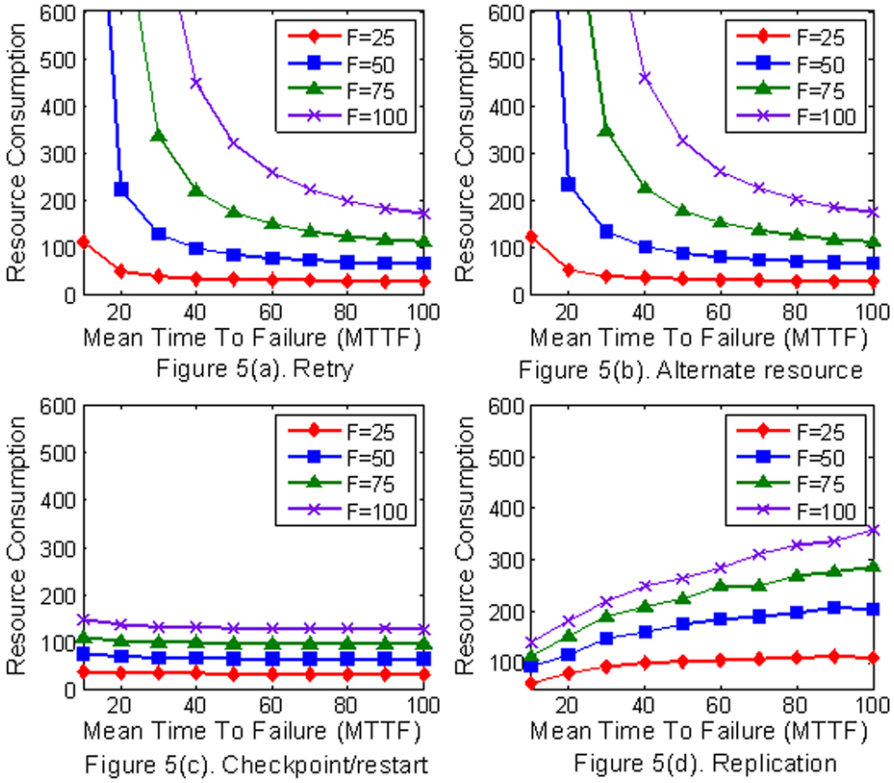


Fig. 5 Resource consumption of the 4 basic approaches (replication parameter N fixed to 5)

6 Experiment

Experiments have been carried out on a real Bioinformatics Grid (BioGrid) constructed with the middleware ChinaGrid Support Platform (CGSP) [21]. BioGrid has 7 nodes distributed in 6 distant cities of China (Beijing, Shanghai, Wuhan, Ji'nan, Lanzhou, Xi'an), and each of them has a cluster with processor number differing from 64 to 256. The example under the experiments is a bioinformatics application program Tigr [19]. Without the loss of generality, we executed Tigr using 4 traditional fault-tolerant approaches and our approach separately, and running 100 times for each in order to discover various fault situations as possible. The program was submitted through different sites. The Tigr we selected needs 40.472 min to complete in a failure-free environment. So the whole experiment lasted over two weeks.

A prototype of the fault-tolerant Grid job manager has been developed. The prototype has been running under the experimental environment with replication parameter $N = 3$. We obtained the real failure occurrence of the whole Grid by CGSV (ChinaGrid Super Vision) [4]. CGSV is the monitoring system along with CGSP. It monitors the status of each cluster's nodes and the jobs submitted to CGSP.

As shown in Table 2, the mean execution time of Tigr using our approach is very close to the time it runs in a failure-free environment. The mean execution time

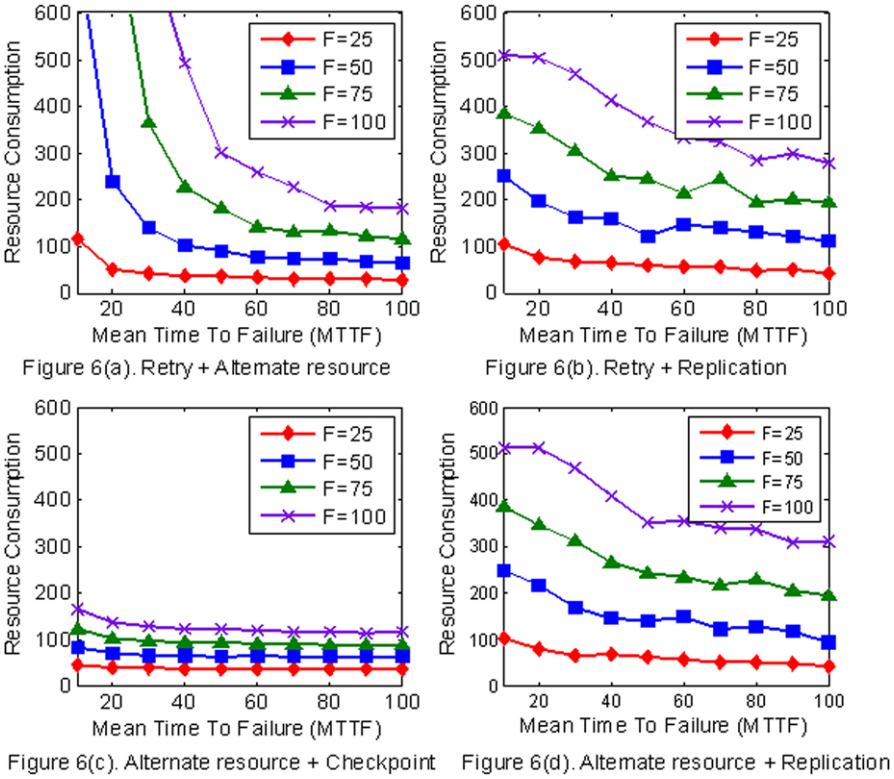


Fig. 6 Resource consumption of combinations of 2 of the four basic approaches (replication parameter N fixed to 5)

Table 2 Mean execution time of Tigr

MTTF (min)	Our approach	Retry	Alternate resource	CK/R	Replication
100	40.88	55.78	54.83	51.03	42.09
150	40.81	51.72	48.78	50.82	42.06
200	40.67	44.92	43.27	42.36	41.02
250	40.66	41.72	41.68	40.92	41.05

is almost constant whenever *MTTF* changes. Retry, alternate resource and checkpoint/restart perform badly when *MTTF* is low. And they are unstable in various situations. In our experiments, they take more than 180 min to finish a 40 min job when *MTTF* is low. Replication also has a very good performance in terms of mean execution time.

Table 3 presents the experiment result of resource consumption of our approach and the four basic approaches. As stated before, resource consumption of retry, alternate resource and checkpoint/restart are the same as their mean execution time. They consume less resource than replication and our approach, because they do not

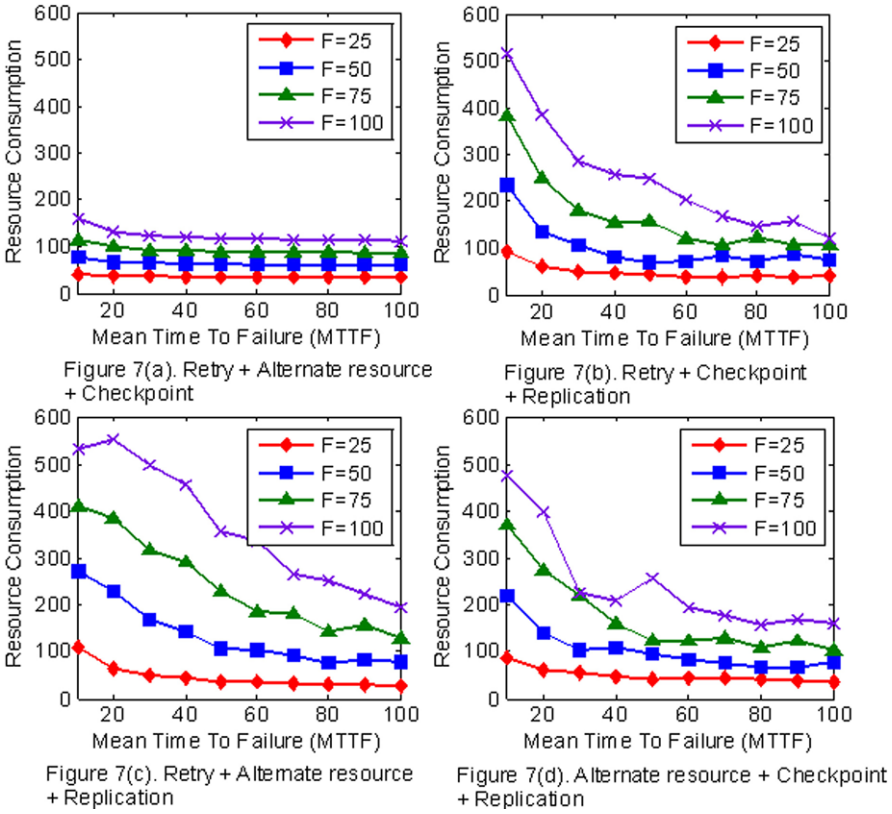


Fig. 7 Resource consumption of combinations of 3 of the four basic approaches (replication parameter N fixed to 5)

Table 3 Resource consumption of Tigr

MTTF (min)	Our approach	Retry	Alternate resource	CK/R	Replication
100	67.42	55.78	54.83	51.03	104.8
150	55.18	51.72	48.78	50.82	110.3
200	48.58	44.92	43.27	42.36	117.1
250	45.02	41.72	41.68	40.92	119.6

need replicated tasks. But they have their inherent shortcomings in terms of mean execution time. Resource consumption of replication increases with the increase of *MTTF*. However, resource consumption of our approach decreases with the decline of *MTTF*.

In other words, our approach can make a program run correctly as replication when the situation is bad (*MTTF* is low), and it can perform as well as retry, alternate resource and checkpoint/restart when things are better. Besides it consumes less resource compared with replication.

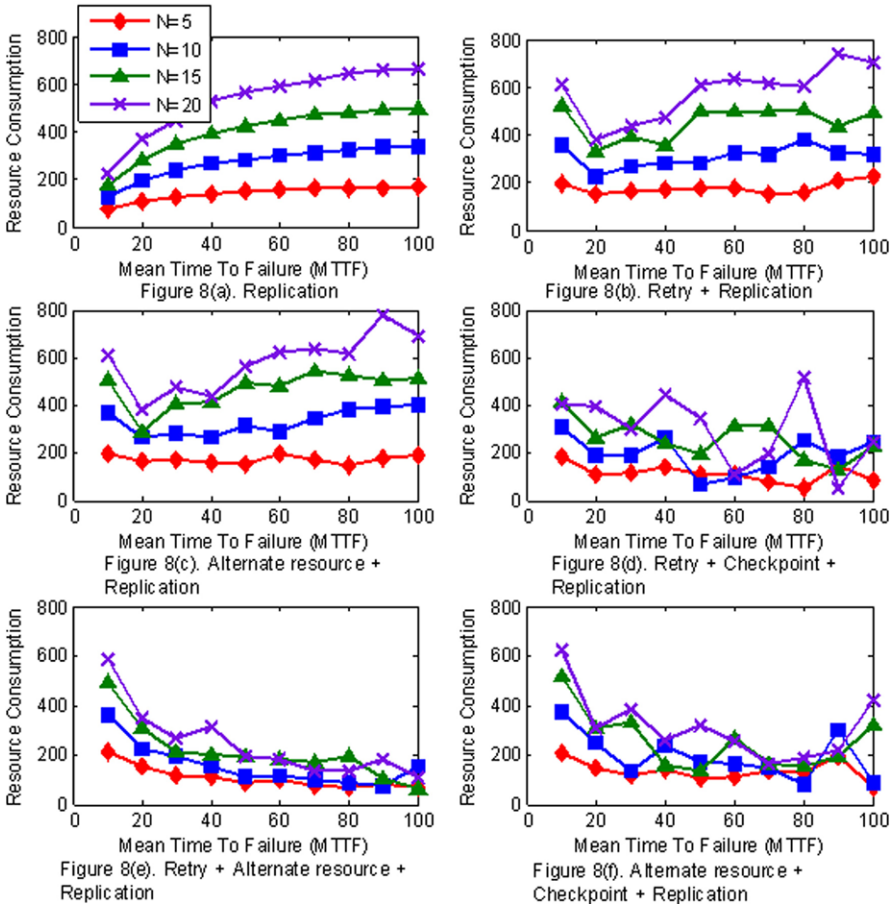


Fig. 8 Resource consumption of replication supported approaches (F fixed to 40)

Consistent conclusion can be driven from the simulation in Sect. 5 and above experimental results for comparison of our approach with the basic four. Based on further deduction and experiment, we also get similar conclusion for the comparison of our approach with the combinations of three or less of the four.

7 Conclusions and future work

Grid is an unreliable, heterogeneous environment and used to provide massive computational capability, but with diverse, complicated failures. Existing methods for fault-tolerant of Grid are not powerful enough to cope with all possible failures in the dynamic Grid environment. Therefore, we proposed an adaptive task-level, fault-tolerant approach which is able to dynamically adapt to and automatically handle diverse failures. An evaluation model (for calculating mean execution time) is proposed and used to evaluate our approach, along with resource consumption which

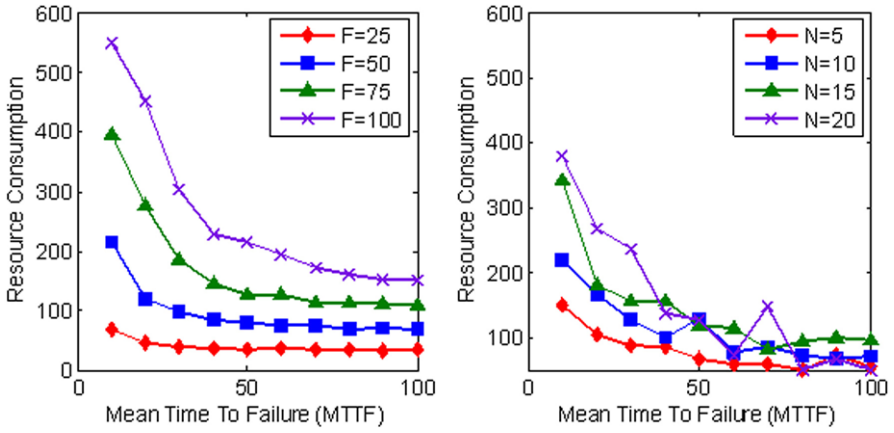


Fig. 9 Resource consumption of our approach

is newly introduced as the other important metric for the evaluation of fault-tolerant approaches. Simulations and a set of experiments are performed on Bioinformatics Grid to evaluate our approach.

Simulation results show that our approach reduces the overhead largely compared with the frequently used four basic fault handling approaches and combinations of two or three of them; the mean execution time of a program using our approach is very close to the time the program runs in a failure-free environment; our approach also consumes less computational resources. In detail, the results show that our approach and replication supported approaches take less mean execution time than retry and alternate resource, and they are all very close to the failure-free situation. However, the data of resource consumption shows that our approach consumes less resource than replication supported approaches. Furthermore, the resource consumption of our approach decreases with the decline of *MTTF*. When Grid environment is bad (i.e., unstable, *MTTF* is low), our approach can dynamically and automatically decides which basic approach to apply based on the analysis for the state of the program under running in order to ensure the complete execution of the program; while when the situation is good (i.e., stable, *MTTF* is high), our approach can complete the program as replication but with competitive performance since in such case retry and alternate can be adaptively applied by our approach. Experiments based on BioGrid also show that our approach can achieve better performance and consumes less resource.

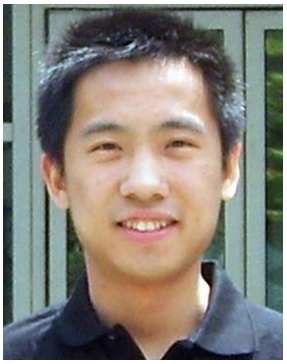
In this work, in terms of the metric resource consumption, only CPU resource consumption is taken care of; however, our approach can also be easily adapted to cover other types of resource consumptions such as memory, I/O, etc. We are also planning to put some attention on Grid-level fault tolerance, especially for those based on multiple different Grid middleware and/or workflow engines.

Acknowledgements This work is co-sponsored by Natural Science Foundation of China (60573110, 90612016, 60673152, 60773145), National High-Tech R&D (863) Program of China (2006AA01A101, 2006AA01A106, 2006AA01A108, 2006AA01A111, 2006AA01A117), and National Basic Research (973) Program of China (2003CB317007, 2004CB318000).

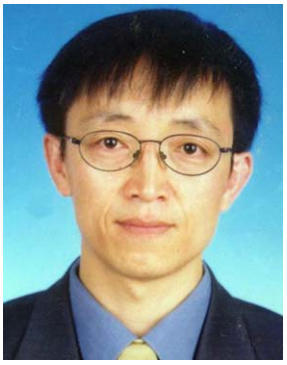
References

1. Beguelin A, Seligman E, Stephan P (1997) Application level fault tolerance in heterogeneous networks of workstations. J Parallel Distrib Comput
2. Cao J et al (2003) GridFlow: workflow management for grid computing. In: CCGrid2003, Tokyo, Japan. IEEE Press, Los Alamitos
3. Casanova H, Dongarra J (1996) NetSolve: a network server for solving computational science problems. In: Proceedings of the ACM/IEEE conference on supercomputing, 1996
4. CGSV project (2008) http://www.chinagrid.edu.cn/CGSV/
5. Deelman E et al (2005) A framework for mapping complex scientific workflows onto distributed systems. Sci Program J 13(3):219–237
6. Duda A (1983) The effects of checkpointing on program execution time. Inf Process Lett 16:221–229
7. Fahringer T et al (2005) ASKALON: a tool set for cluster and grid computing. Concurr Comput Pract Exp 17:143–169
8. Frey J, Tannenbaum T, Foster I, Livny M, Tuecke S (2002) Condor-G: a computation management agent for multi-institutional grids. Cluster Comput
9. Hwang S, Kesselman C (2003) Grid workflow: a flexible failure handling framework for the grid. In: 12th IEEE international symposium on high performance distributed computing (HPDC'03), 2003
10. Jin H, (2004) ChinaGrid: making grid computing a reality. In: Digital libraries: international collaboration and cross-fertilization. LNCS, vol 3334. Springer, Berlin, pp 13–24
11. Li K, Shen H (2005) Coordinated enroute multimedia object caching in transcoding proxies for tree networks. ACM Trans Multimed Comput Commun Appl (TOMCAPP) 5(3):289–314
12. Li K, Shen H, Chin FYL, Zhang W (2007) Multimedia object placement for transparent data replication. IEEE Trans Parallel Distrib Syst 18(2):212–224
13. Li K, Shen H, Chin FYL, Zheng SQ (2005) Optimal methods for coordinated enroute web caching for tree networks. ACM Trans Internet Technol (TOIT) 5(3):480–507
14. Oinn T et al (2004) Taverna: a tool for the composition and enactment of bioinformatics workflows. Bioinformatics 20(17):3045–3054
15. Plank JS, Elwasif WR (1998) Experimental assessment of workstation failures and their impact on checkpointing systems. In: Proceedings of the 28th fault-tolerant computing symposium, 1998
16. Stelling P, Foster I, Kesselman C, Lee C, von Laszewski G (1998) A fault detection service for wide area distributed computations. In: Proceedings of the seventh IEEE symposium on high performance distributed computing, pp 268–278, 1998
17. Tannenbaum T, Wright D, Miller K, Livny M (2002) Condor—a distributed job scheduler. In: Beowulf cluster computing with Linux. The MIT Press, Cambridge
18. Taylor I, Shields M, Wang I (2003) Resource management of Triana P2P services. In: Grid resource management. Kluwer, Dordrecht
19. Tigr (2008) http://www.tigr.org
20. von Laszewski G (2005) Java CoG kit workflow concepts for scientific experiments. Technical Report, Argonne National Laboratory, Argonne, IL, USA
21. Wu Y, Wu S, Yu H et al (2005) CGSP: an extensible and reconfigurable grid framework. 2005:292–300. Springer (2005)
22. Yu J, Buyya R (2004) A novel architecture for realizing grid workflow using tuple spaces. In: 5th IEEE/ACM international workshop on grid computing (GRID 2004), 2004

Yongwei Wu received his Ph.D. degree in the Applied Mathematics from the Chinese Academy of Sciences in 2002. Since then, he has worked at the Department of Computer Science and Technology, Tsinghua University, as research Assistant Professor from 2002 to 2005, and Associate Professor since 2005. His research interests include grid and cloud computing, distributed processing, and parallel computing. He is a member of the IEEE.



Yulai Yuan received his B.Sc. degree in Computer Science and Engineering from the Beijing Information Technology Institute, Beijing in 2005. He is currently pursuing his Ph.D. degree in Computer Science at Tsinghua University. His research interests include distributed systems, performance modeling and prediction, and data replication.



Guangwen Yang is Professor of Computer Science and Technology, Tsinghua University, China. He is also an expert on "high-performance computer and its core software", the National "863" Program of China. He is Deputy Director of the Academic Committee of China Computer Society, an IEEE member, and a member of National Library Expert Advisory Committee. He is mainly engaged in the research of grid computing, parallel and distributed processing and algorithm design and analysis.

Weimin Zheng is Professor of Computer Science and Technology, Tsinghua University, China. He received the B.Sc. and M.Sc. degrees form Department of Automatics, Tsinghua University, in 1970 and 1982 respectively. Currently he is the research director of Institute of High Performance Computing, Tsinghua University, and managing director of the Chinese Computer Society. His research interests include computer architecture, operating system, storage networks, and distributed computing.