A Decentralized Fault Tolerant model for Grid Computing

Mohammed REBBAH[1], Yahya SLIMANI[2], Abdelkader BENYETTOU[1]

[1] Université des Sciences et Technologie d'Oran – Mohammed BOUDIAF, ALGERIE

[2] Computer Science Department, University of El Manar, Tunis, Tunisia

Abstract

A current trend in high-performance computing is the use of large-scale computing grids. These platforms consist of geographically distributed cluster federations gathering thousands of nodes. At this scale, node and network failures are no more exceptions, but belong to the normal system behavior. Thus, grid applications must tolerate failures and their evaluation should take reaction to failures into account. The failures of distributed computing system can be divided into three categories: node crash, network failure and process fault. The fault tolerance is a significant and complex issue in grid computing systems. Various techniques have been investigated to detect and tolerate faults in distributed computing systems. We propose, in this paper, a decentralized model of fault tolerance based on dynamic colored graphs. From this model, we show through some experiments, the benefits of colored graphs to manage failures in grids.

Keywords: *Large scale systems, Grid computing, Fault tolerance, Dynamic colored graph.*

1. Introduction

A current trend in high-performance computing is the use of large-scale computing grids. These platforms consist of geographically distributed cluster federations gathering thousands of nodes on the Internet [1]. At this scale, node and network failures are no more exceptions, but belong to the normal system behavior. Thus, grid applications must be able to tolerate failures to ensure some quality of service in grids. The failures of distributed computing system can be divided into three categories [2]: node crash, network failure and process fault. The fault tolerance is a significant and complex issue in grid computing systems. Various techniques have been investigated to detect and correct faults in distributed computing systems [2]. In this paper, we address the problem of fault tolerance in grids by proposing two complementary models: the first model is used to represent a grid as a dynamic colored graph. Starting from this representation, the second model defines a fault tolerance technique. The main contribution of the proposed fault tolerance model is to define for each node in a grid, a set of neighbors collaborators able to replace it in case of failure. The number of neighbors collaborators of a node is limited by a threshold α defined by the user. In a first phase, we identify collaborators from neighboring nodes (nodes having a direct connection with the failed one). These nodes are then classified into three categories depending on the value of the threshold α as follow: stable, unstable and hyperstable nodes. In a second phase, called stabilization phase, each unstable node attempts to auto stabilize through hyperstable nodes across the graph.

The reminder of the paper is organized as follows: Section 2 gives an overview on different works on fault tolerance in grid computing. Related work is discussed in section 3. Section 4 describes our proposition for modeling a grid with a dynamic colored graph; it also discusses issues vertex coloring, the computation of the tolerance threshold, the graph stabilization and the different types of dynamicity. Section 5 presents a mathematical formulation of the proposed model. Section 6 discusses some experimental results of our model. Finally, Section 7 concludes the paper and gives some directions for future research.

2. Fault tolerance in grid computing

Failure in large-scale Grid systems is and will be a fact of life. Hosts, networks, disks and applications frequently fail, restart, disappear and behave unexpectedly. Support for the development of fault tolerant applications has been identified as one of the major technical challenges to address for the successful deployment of computational grids [3, 4, 5]. Three techniques for fault tolerance in grid computing have been of particular importance: (i) checkpointing, or periodical saving of the state of a process running on a computational resource so that, in the event of failure, it can be migrated to an operational resource [6,7], (ii) replication, i.e. maintaining a sufficient number of replicas, or copies, of a process executing in parallel with identical state but on different resources, that at least one replica is almost guaranteed to finish the process correctly [8,9,10] and (iii) in the event of failure, rescheduling, or finding different resources to that can accept and run failed tasks [11,12,13]. Several approaches for the implementation of fault tolerance in message-passing applications exist. MPICHGF [14] is a checkpointing system based on MPICH-G2 [15], a Grid-enabled version of MPICH. It handles checkpointing, error detection, and process restart in a manner transparent to the user [16]. Pawel Garbacki et al. address the problem of making java-RMI parallel applications fault tolerant in a way transparent to the programmer [17]. Azzedin and Maheswaran [18] suggest to integrate the concept of trust into grid resource management. Abawajy [19] present a Distributed Fault-Tolerant Scheduling System (DFTS) to provide fault tolerance for jobs execution in a grid environment. Song [20] developed a security-binding scheme through site reputation assessment and trust integration across grid sites. A Fuzzy-logic based Self-Adaptive job Replication Scheduling (FSARS) algorithm is proposed to handle the fuzziness or uncertainties of job replication number which is highly related to trust factors behind grid sites or user jobs was presented by Congfeng Jiang et al [21].

3. Related work

A Grid system architecture describes the structure of a grid system, which consists of nodes and their interconnections. Nodes, or sites, contain grid resources and related software components. A Grid architecture can be viewed as the high-level design of a grid system. It is very important with regards to scalability, autonomy, and performance of the system. It can be divided into three categories: centralized, hierarchical, and decentralized. In contrast, decentralized organization negates the limitations of centralized or hierarchical organization with respect to fault-tolerance, scalability, and autonomy (facilitating domain specific resource allocation policies). This approach scales well. However, this approach raises some challenges related to distributed information management, system-wide coordination, security, and resource provider's policy heterogeneity. For instance, the complexity of a decentralized Grid system is much higher than a centralized Grid system because of the interaction and coordination of a large number of decentralized components.
A number of researchers investigated how decentralizing data replication management services can be used to improve fault tolerance in data grids. In [22], Chervenak et al. described a decentralized replica location service for the Globus toolkit [23] that was designed to be scalable and fault tolerant. Here, distributed, redundant indexes maintaine information on data replicas in a consistent manner. Zhang et al. [24] proposed an algorithm for dynamically locating data replica servers within a grid to optimize performance and improve fault tolerance, though scalability issues were not examined. In [25], Abbes et al. proposed to eliminate the need for a centralized server, therefore to remove the single point of failure and bottleneck of existing Desktop Grids. Instead, each node can play alternatively the role of client or server. They designed the PastryGrid protocol (based on Pastry) for Desktop Grid in order to support a wider class of applications, especially for distributed applications with precedence between tasks. In [26], Aliaa et al. describes an agent based resource management system, ARMS, that includes an adaptable decentralized service advertisement strategy to reduce the cost of the advertisement process within ARMS system. A Performance Monitoring and Advisory component is implemented to compare the developed strategy with those previously defined in the literature. This strategy is more adaptable to continuous changes in both workload and structure.
All these works did not include the identification of nodes that can replace each failed node in the grid. In this paper, each node identifies a set of neighbors collaborators able to replace it in case of failure. The number of neighbors collaborators of a node is limited by a threshold α. The nodes are then classified into three categories depending on the value of the threshold α as follow: stable, unstable and hyperstable nodes.
Much of the works that model the grid as a graph [27, 28, 29], where the vertices represent the nodes of the grid and connections between nodes by edges, but they do not model the dynamic nature of grid resources, that can unpredictably appear and disappear. In this paper, we propose a decentralized fault tolerance model; we take into account all its aspects by modeling the grid as a dynamic colored graph.

4. Model graph

In this paper, we model a grid computing as an undirected graph $G=(V, E)$, where the vertices $V=\{v_1,v_2,...,v_n\}$ represent all the resources (nodes) of the grid; the set $E=\{e_1,e_2,\ldots, e_m\}$ represents the edges (connections) between the resources. Also, each edge $e_i \in E$ has an arbitrary, non-negative weight. The distance between two vertices v_i and v_j, $d(v_i,v_j)$, is the sum of the weights of edges along a shortest path between v_i and v_i. To take into account the dynamic nature of grid resources, the graph G will be dynamic: at any time t, the graph may be subject to additions and/or deletions of vertices or edges. For this, we define as $G_t=(V,E)$, the graph G at time t. The function $Neighbor_t(v_i)$ denotes all neighboring vertices of vertex v_i at time t. To take into account the heterogeneity of resources, we associate with each vertex a color indicating the type of the resource, such that all the colors represent the types of resources in the grid (see Definition 1). By combining all these properties, we can model a grid as a dynamic colored graph (see Definition 2).

Definition 1: (COLORED GRAPH)

A colored graph $G = (V, E, C, A)$ is a 4-uplet defined as follows:

V: non-empty finite set representing the vertices
E: $V \rightarrow V$; set of edges
C: non-empty set of colors

A: $V \rightarrow C$; set of colors assigned to each vertex.

Definition 2: (DYNAMIC COLORED GRAPH)

A dynamic graph colored ξ is the triple (G_0, T, P) defined as follows:

G_0=(V, E, C, A) is the initial colored graph (at time 0)
T is a continuous or discrete time basis
P is an evolutionary process.

4.1. Coloring rule

Each vertex v_i of the graph consults its neighbors *$Neighbor_t(v_i)$* for mutual cooperation in case of failure; these vertices are considered as neighbors collaborators **(*$NeighbCol_t(v_i)$*).** We define, for a graph, a threshold α that determines the sufficient number of neighbors to adequately tolerate the faults of any vertex v_i; this threshold represents the degree of tolerance in our model. *$C_t(v_i)$* defines the color of the vertex v_i at time *t*. For each vertex v_i, if **|*$NeighbCol_t(v_i)$*|=α,** it will be colored in Green color *($C_t(v_i)$= G)* and it will be considered as a **stable vertex**; if **|*$NeighbCol_t(v_i)$*|<α,** it will be colored in Red color (*$C_t(v_i)$*= R) and it will be considered as **unstable vertex**; finally if **|*$NeighbCol_t(v_i)$*|>α**, it will be colored in Blue color (*$C_t(v_i)$*= B) and it will be considered as **hyperstable vertex** (see Figure 1).

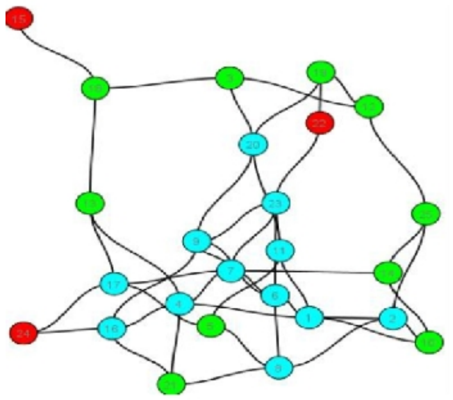
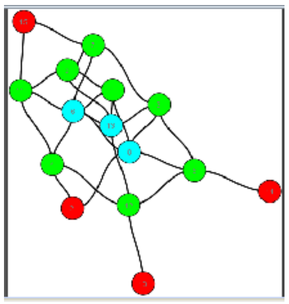


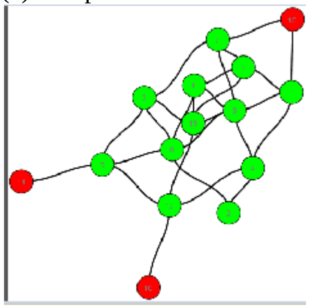
Fig. 1: Dynamic colored graph (25 vertices, 45 edges and α =3).

4.2. Stabilization protocol

Once the vertices are colored, the unstable vertices (with Red color) attempt to auto stabilize through hyperstable vertices. A vertex v_i with color *$C_t(v_i)$*=R explores *$Neighbor_t(v_i)$* while looking for hyperstable vertices, e.g. *$Hyp_t(v_i)$*={*y*| *$C_t(y)$*=B and *$y \in Neighbor_t(v_i)$*}. The vertex v_i sends requests to vertices *$y \in Hyp_t(v_i)$* to provide it, among their Neighbors Collaborators {*z*| *z*∈***$NeighbCol_t(y)$***, *$z \notin Neighbor_t(v_i)$ and $z \in Hyp_t(y)$*}, e.g., the vertices that accept to collaborate with v_i. This will reduce the number of ***$NeighbCol_t(y)$*** of the vertex *y* until α (e.g., *y* becomes stable). For the vertex v_i, the process stops when it becomes stable. Otherwise, the vertex v_i looks in the graph for the closest hyperstable vertex to stabilize. The stabilization process stops when it's impossible to change anymore the vertices color in the graph (see Figure 2).



(a): Graph G after coloration



(b): Graph G after stabilization

Fig. 2: Graph G (15 vertices, 27 edges and α=4) after coloration and stabilization

4.3. Graph dynamicity

The dynamicity of the graph is characterized by the addition and the deletion of edges and vertices. Foremost, we must ensure the presence of vertices. To do so, we define a delay t_{max} where each vertex must send its heartbeat message to all its neighbors; after this delay, it will be considered absent from the graph and its neighbors are responsible for transmitting its failure to all other vertices of the graph. When a new vertex is added to the graph, it will be colored according to the number of its neighbors and then it will undergo a possible stabilization phase. When the graph reaches a high level of dynamicity, we reactivate the coloration and the stabilization in the whole graph.

5. Mathematical formulation

In this section, we propose a mathematical formalization of our proposed model.

5.1. Vertices state

The vertices are colored during the phase of coloration by the three basic colors (Red, Green and Blue). During the stabilization phase, the vertices may eventually change color following the lemmas 1 and 2, where each red vertex tent to stabilize while consuming the surplus of the neighboring collaborators of the blue vertices.

Lemma 1:
Let v_i be a vertex such that $C_t(v_i)=Red$ **and** $|NeighbCol_t(v_i)|=\beta$ ($\beta<\alpha$). Vertex v_i can change its color to Green if and only if $\exists$ a set $A=\{y|\ C_t(y)=Blue$ **and** $\sum|NeighbColt(y)|=\alpha-\beta\}$.

Proof: Consider a vertex v_i with a set of neighbors collaborators less than α. The change of its color is required by the existence of one or more vertices having neighbor collaborators above the threshold α and which can complete the neighbor's collaborators of v_i until it reaches α. In this case, its color will become Green otherwise it remains Red (see Figure 3).

Lemma 2:
Let v_i be a vertex such that $C_t(v_i)=Blue$ and $|NeighbCol_t(v_i)|=\beta$ ($\beta>\alpha$) . Vertex v_i can change its color to Green if and only if $\exists$ a set $A=\{y/\ C_t(y)=Red$ **and**$\sum|NeighbColt(y)|=\beta-\alpha\}$.

Proof: Consider a vertex v_i with a set of neighbors collaborators above the threshold α. The change of its color is determined by the existence of one or more vertices having neighbors collaborators less than α and which can consume the neighbors collaborators of v_i until it reaches α and it color will become Green, otherwise it remains Blue (see Figure 3).

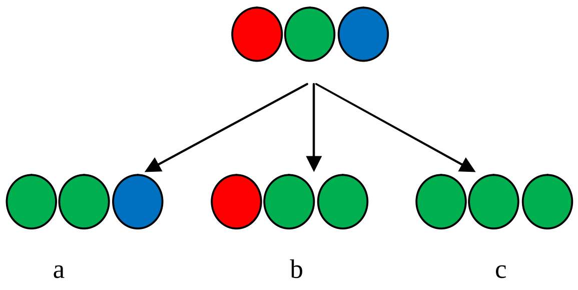


Fig.3: Different states of the graph vertices

The Figure 3 shows the different situations when vertices change its colors:

a) In Figure 3(a), the unstable vertex (Red color) stabilizes (becomes Green) throught the hyperstable vertex, which remains hyperstable.

b) In Figure 3(b), the unstable vertex fails to stabilize although it took all neighboring collaborators of the hyperstable vertex, which become stable.

c) In Figure 3(c), the unstable vertex stabilizes with all neighboring collaborators of the hyperstable vertex and they become all stables.

5.2. Graph state

The phases of coloration and stabilization generate various kinds of colored graphs. Also, we will focus on all the colors that might exist in the graph at time t.

Definition 3:
The state of the graph G, defined as ($State_t$ (G)), is a set of the current colors in the graph at time t.

$State_t$ (G) = {GB} means that all the colors present in G are Green and Blue (see Figure 4).

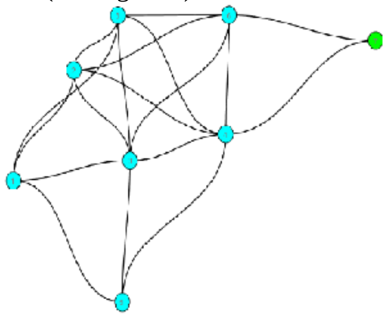


Fig. 4: Dynamic colored graph (8 vertices, 18 edges and α = 2); $State_t$ (G) = {GB}.

From the three basic colors (RGB), all possible states of the graph are: $State_t$ (G) = {G, R, B, GR, GB, RB, RGB}. The Table 1 shows each state and its phase of appearance.

State	Coloration	Stabilization
R	Yes	Yes
B	Yes	Yes
G	Yes	Yes
GB	Yes	Yes
GR	Yes	Yes
RB	Yes	No
RGB	Yes	No

Table 1: Different states of the dynamic colored graph and their corresponding phases.

Three cases are possible:

1. *$State_t$ (G) = {R} or {B}*: these two states can be generated only from the coloration phase and are stable directly; we will call them **isolated states**.

2. *$State_t$ (G) = {G} or {GB} or {GR}*: these three states can be obtained from the coloration phase, or from the states of the 3rd case, and they are also stable; we will call them **terminal states**.

3. *$State_t$ (G) = {RB} or {RGB}:* these two states are only obtained from the coloration phase and they represent the **intermediate states** that always converge toward a terminal state; the state *{RB}* converges toward the states *{G} or {GB}* and the state *{RGB}* converges toward all the terminal states *{G} or {GB} or {GR}* (see Figure 5).

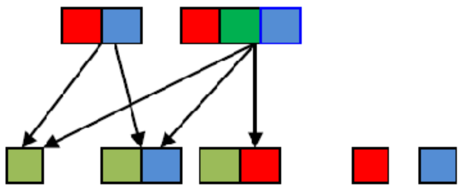


Fig. 5: Different graph states

Theorem: The graph always converges to a stable state *{G, R, B, GR, GB}*.

Proof: For each graph *G*, we have *$State_t$ (G) ={G, R, B, GR, GB, RB, RGB}*; according to Lemmas 1 and 2, only the Blue and Red vertices can change their colors. The vertices continue to change their colors if there is no simultaneous

Red and Blue vertex; therefore, *G* always converges to the isolated and terminal states.

6. Evaluation and performance

In this section, we evaluate the performance of our model through a simulation based on discrete events, using Graphstream [33]. We measure in particular, its transient behavior (the number of different colors of the vertices according to α, the stabilization time, the influence of computation of the tolerance degree α on the coloration and stabilization of vertices, the scalability of the model and the impact of the rate of dynamicity on the structure of the graph). Each simulation is performed on an undirected graph, randomly generated as follows: given a set of vertices and a number of edges added between vertices pairs, the weights of the edges are selected randomly.

6.1. Computation of the tolerance degree

In our model, the value of the tolerance degree α is crucial for the vertices coloring. We propose four methods to compute the threshold α:

1. **Average neighbors:** In this method, the threshold is calculated using the equation 1.

$$\alpha = \frac{\sum_{vi \in V} Neighbor(vi)}{|V|} \quad (1)$$

2. **Modulo:** With this method, we select the most repeated number of neighbors for all the vertices of the graph.
3. **Iterative:** In this method, the threshold is calculated iteratively. First, it takes the greatest number of neighbors in the graph, ($\alpha = Max(Neighbor(v_i)), \forall v_i \in V$); then, it is reduced until the number of stable vertices becomes greater than the unstable vertices number.
4. **User:** In this last method, the user defines its own value of *α*.

6.2. Influence of the tolerance degree on the vertices colors

6.2.1. Coloration phase

In this experimentation, we vary the number of vertices from 100 to 1000, we fix the number of edges at 3000, we create a graph for each case and we calculate the colors of vertices using each of the previous methods of the tolerance degree computation (see Table 2).

# Vertices	M1: Average neighbors				M2: Modulo				M3 : Iterative			
	α	R	G	B	α	R	G	B	α	R	G	B
100	60	45	5	50	58	28	10	62	47	0	1	99
200	30	96	16	88	29	79	17	104	22	5	8	187
300	20	140	29	131	20	140	29	131	10	0	3	297
400	15	188	47	165	14	141	47	212	8	4	10	386
500	12	225	66	209	12	225	66	209	6	11	12	477
600	10	277	76	247	8	112	88	400	5	16	23	561
700	8	262	89	349	9	351	99	250	4	26	24	650
800	7	301	114	385	6	178	123	499	4	47	56	697
900	6	316	139	445	6	316	139	445	4	87	95	718
1000	6	464	142	394	5	294	170	536	3	54	95	851

Table 2: Vertex coloring by the three methods of the tolerance degree computation

The iterative method always gives a lower degree of tolerance compared to other methods; colors Red and Green are very close with an average value of 3.30% for Red, 4.52% for Green and 92.18% for Blue. The degrees of tolerance of method M1 are always greater or equal to those of method M2 with a maximum difference equal to 2. Both methods give very small green vertices compared to other colors. The distribution of the Red and Blue colors is not regular.

6.2.2. Stabilization phase

After the stabilization phase of previously colored graphs, all vertices colored by the method M1 converge to the $State_t$ *(G)={G}*; in the contrary, those colored by methods M2 and M3 converge to the $State_t$ *(G)={GB}*, and the Blue vertices reach up to 96.12% in the method M3. The method M1 always gives the best value of the tolerance degree because all vertices converge to a stable state (see Table 3).

# Vertices	M1: Average neighbors				M2: Modulo				M3 : Iterative			
	α	R	G	B	α	R	G	B	A	R	G	B
100	60	0	100	0	58	0	53	47	47	0	1	99
200	30	0	200	0	29	0	151	49	22	0	13	187
300	20	0	300	0	20	0	300	0	10	0	3	297
400	15	0	400	0	14	0	274	126	8	0	14	386
500	12	0	500	0	12	0	500	0	6	0	23	477
600	10	0	600	0	8	0	242	358	5	0	40	560
700	8	0	700	0	9	107	593	0	4	0	51	649
800	7	0	800	0	6	0	383	417	4	0	111	689
900	6	0	900	0	6	0	656	244	4	0	201	699
1000	6	0	1000	0	5	0	604	396	3	0	160	840

Table 3: Stabilization phase by different methods of the tolerance degree computation

6.3. Path length stabilization

The stabilization of a vertex v_i tries to find the maximum of neighbors collaborators in the whole graph. We study the path length from the vertex v_i to its neighbors collaborators; we consider the *length* of a path as the number of edges that the path uses. We vary the number of vertices from 100 to 1000, we fix the number of edges at 3000 and we use the average neighbors method to compute α (M1 method). Table 4 shows the unstable vertices from the coloration phase and the stabilized vertices for each path length. The results show that the maximum length reached is equal to 4, with an average of 87.72% for paths of length 2, 11.35% for paths of length 3 and 0.93% for paths of length 4.

6.4. Stabilization time

In this section, we focus on stabilization time; we varied the number of vertices from 100 to 900 with 3000 edges. The coloring phase offers a varied number of unstable and hyperstable vertices (see Table 2). The stabilization time does not depend on the number of edges in the graph but on the number of unstable vertices that seeks to stabilize and on the number of hyperstable vertices available in the graph.
1. Method M3 provides a very small number of unstable vertices (3.30% of unstable vertices) with a very high number of hyperstable vertices (92.18%), which ensures stability in record time.
2. Methods M1 and M2 provide a very close rate of hyperstable and unstable vertices (43.44% of unstable vertices and 44.87% of hyperstable vertices for M1, and 35.00% of unstable vertices and 52.03% of hyperstable vertices for M2). Their stabilization time is almost the same (see Figure 6).
The stabilization time is the order of milliseconds (5.19 ms for 100 vertices and 14.55 ms for 900 vertices), because of its decentralization, which shows its speed, and the difference of the stabilization time between a graph of 100 vertices and a graph of 900 vertices is 9.35 ms, which shows the scalability of our model (see Figure 6).

7. Conclusion

In this paper, we defined a model transforming a grid computing to a dynamic colored graph. Starting from this graph, we proposed a mechanism of fault tolerance. For the coloration of vertices (nodes of a grid), we defined a degree of tolerance, noted α, allowing us to obtain unstable, stable and hyperstable vertices. This characteristic of the vertices depends, for a given vertex, on the number of its neighbors collaborators (vertices that can replace it in case of failure). Regarding the unstable vertices, we defined a stabilization process that allows the unstable vertices to stabilize through the hyperstable ones. In the proposed model, the graph always converges to a stable state. The simulation results show the influence of the tolerance degree computation method in the convergence of the graph. We noted from our experimentations that the method based on the average neighbors (method M1) yielded better results in terms of stabilization of vertices. In this method, the graph converges rapidly with little effect of the number of vertices of the graph, which makes it very effective in scalability. Based on these preliminary results, it seems that the proposed model can be used for other environments like grids, where we can found dynamicity, heterogeneity and scalability. Among these environments, we think that P2P systems and wireless networks provide good platforms for testing the adaptability of our model to this type of infrastructure. We also believe that the proposed model can be adapted to address other issues of large-scale systems, such as load balancing and data replication.

# Vertices	# Unstable vertices	# Stable vertices		
		Path length= 2	Path length= 3	Path length= 4
100	45	44	1	0
200	96	89	7	0
300	140	126	13	1
400	188	173	14	1
500	255	185	66	4
600	277	227	44	6
700	262	228	30	4
800	301	253	41	7
900	316	270	33	13
1000	464	395	50	19

Table 4: Stable vertices by path length

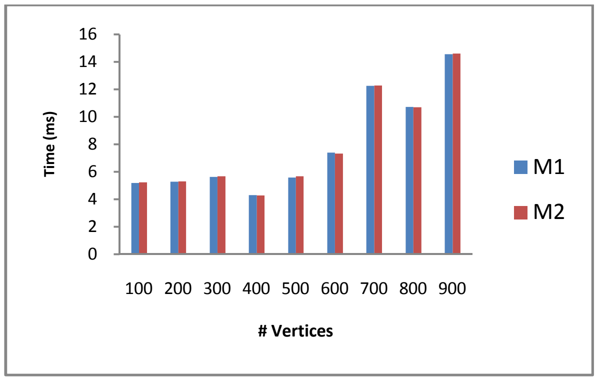


Fig. 6: Stabilization time

References

[1] Foster, I., Kesselman, C.: *The Grid: Blueprint for a New Computing Infrastructure.* eds. San Francisco, Calif.: Morgan Kaufmann Publishers, 1999, 677 P.

[2] Jin, H., Zou, D., Chen, H., Sun, J., Wu, S., : Fault-Tolerant Grid Architecture and Practice. *Journal of Computer Science and Technology* 18(4): 423-433 (2003).

[3] Garg, R., Singh, A. K.: Fault tolerance grid computing: state of the art and open issues. International Journal of Computer Science & Engineering Survey (IJCSES) Vol.2, No.1: 88-97, Feb 2011.

[4] Siva Sathya, S., Syam Babu, K.: Survey of fault tolerant techniques for grid, *Computer Science Review*, Vol.4, No. 2: 101-120, 2010.

[5] Anju Bala, Inderveer Chana Fault Tolerance- Challenges, Techniques and Implementation in Cloud Computing, International Journal of Computer Science Issues, Vol. 9-1, pp. 288-293, 2012

[6] Jin, H., Shi, X., Qiang W. and Zou., D. : DRIC: Dependable Grid Computing Framework,, *IEICE - Transactions on Information and Systems*, Vol.E89-D, No.2:612-623, February 2006 [doi>10.1093/ietisy/e89-d.2.612].

[7] Jafar, S., Krings, A., and Gautier, T.: Flexible Rollback Recovery in Dynamic Heterogeneous Grid Computing, *IEEE Transactions on Dependable and Secure Computing*, Vol. 6, No. 1:32-44, JANUARY-MARCH 2009

[8] Lac C, Ramanathan S. A Resilient Telco Grid Middleware. *Proceeding of 11th IEEE Symposium on Computers and Communications (ISCC'06)*, June 2006. IEEE Computer Society Press: Los Alamitos, CA, pp. 306-311, 2006.

[9] Jiang. C., Xu. X., Wan, J.: Replication Based Job Scheduling in Grids with Security Assurance, *Proceedings of the Third International Symposium on Electronic Commerce and Security Workshops (ISECS '10)* Guangzhou, P. R. China, 29-31, pp. 156-159, July 2010.

[10] Sangho, Y.,, Derrick, K., Bongjae, K., Geunyoung, P., Yookun, C.: Using Replication and Checkpointing for Reliable Task Management in Computational Grids, *International Conference on High Performance Computing and Simulation (HPCS)*, pp 125 - 131 , France 2010

[11] Radha, V.Sumathy A Detailed Study of Resource scheduling and Fault Tolerance in Grid, International Journal of Computer Science Issues, Vol. 8-6,pp. 357-361, 2011

[12] OLTEANU, A., POP, F., DOBRE, C., CRISTEA, C.: Re-scheduling and error recovering algorithm for distributed environments, *U.P.B. Scientific Bulletin*, Series C, Vol. 73, Iss. 1: 27-38, 2011.

[13] Leyli, M. K., Maryam E. F., Ali G., Reliable Job Scheduler using RFOH in Grid Computing, *Journal of Emerging Trends in Computing and Information Sciences* Vol. 1, No. 1:43-48, July 2010

[14] Woo, N., Jung, H., Yeom, H. Y., Park, T. and Park., H.: MPICHGF: Transparent Checkpointing and Rollback-Recovery for Grid-Enabled MPI Processes. *IEICE Transactions on Information and Systems*, 87(7):1820–1828, 2004.

[15] Nicholas, I. F., Karonis, T., Toonen, B: MPICH-G2: A Grid-enabled implementation of the Message Passing Interface. *Journal of Parallel and Distributed Computing*, 63(5):551–563, May 2003.

[16] Díaz, D., Pardo, X. C., Martín, M. J., González, P.: Application-Level Fault-Tolerance Solutions for Grid Computing; *Eighth IEEE International Symposium on Cluster Computing and the Grid (CCGRID'08)*, IEEE Computer Society, Washington, USA, pp. 554-559, 2008.

[17] Pawel, G., Bartosz, B., Henri, E. B.: Transparent Fault Tolerance for Grid Applications. *Proceedings of the European Grid Conference (EGC 2005),* Amsterdam, The Netherlands, pp. 671-680, 2005.
[18] Azzedin, F., Maheswaran, M.: Integrating trust into grid resource management systems; *In: Proceedings of the International Conference on Parallel Processing (ICPP'02),* IEEE Computer Society Press, Los Alamitos, pp. 47–54, 2002.
[19] Abawajy, J.: Fault-Tolerant Scheduling Policy for Grid Computing Systems; *In Proceedings of the 18th International Parallel and Distributed Processing Symposium, IPDPS'04,* Santa Fe, New Mexico, pp. 238–244, 2004.
[20] Song, S., Hwang, K., Kwok, Y.: Trusted grid computing with security binding and trust integration; *Journal of Grid Computing* , pp. 53–73, 2005.
[21] *Jiang, C*., Wang, *C*., Liu, X., Zhao, Y.: A Fuzzy Logic Approach for Secure and Fault Tolerant Grid Job Scheduling; *Autonomic and Trusted Computing, 4th International Conference, ATC 2007*, Hong Kong, China, Volume 4610, pp. 549-558 of Lecture Notes in Computer Science, Springer, July 11-13, 2007.
[22] Chervenak, A., Palavalli, N., Bharathi, S., Kesselman, C?, Schwartzkopf, R.: Performance and scalability of a replica location service. *Proceedings of the 13th IEEE International Symposium on High Performance Distributed Computing,* June 2004. IEEE Computer Society Press: Los Alamitos, CA, 182–191, 2004.
[23] The Globus Toolkit. http://www.globus.org/toolkit/ [May, 20, 2011].
[24] Zhang, Q., Yang, J., Gu, N., Zong, Y., Ding, Z., Zhang, S.: Dynamic replica location service supporting data Grid systems. *Proceedings of the Sixth IEEE International Conference on Computer and Information Technology*, IEEE Computer Society Press: Los Alamitos, CA, September 2006.
[25] Abbes, H., Crin, C.: A decentralized and fault-tolerant desktop grid system for distributed applications, *Concurrency and Computation: Practice and Experience*; 22(3):261–277, 2010.
[26] Aliaa, A. A. Y., Atef, Z. G., Mohammed, E. E. D.: An Efficient Decentralized Grid Service Advertisement Approach Using Multi-Agent System. *Computer and Information Science*, Vol. 3, No.:2: 220-228, 2010.
[27] Kumar, S., Das ,S. K., Biswas, R.: Graph Partitioning for Parallel Applications in Heterogeneous Grid Environments, *Proceedings of the 16th International Parallel and Distributed Processing Symposium,* p.167, April 15-19, 2002
[28] Pallis, G., Katsifodimos, A., Dikaiakos, M.D.: Searching for Software on the EGEE Infrastructure. *Journal of Grid Computing*, Vol. 8, No.2: 281-304, 2010.
[29] Bissias, G.D., Levine, B.N., Sitaraman, R.K.: Assessing the vulnerability of replicated network services. *7th International Conference on emerging Networking EXperiments and Technologies (CoNEXT)*, November, 2010.
[33] Dutot, A., Guinand, F., Olivier, D., Pigné, Y.: Graphstream: A tool for bridging the gap between complex systems and dynamic graphs; *Emergent Properties in Natural and Artificial Complex Systems. Satellite Conference within the 4th European Conference on Complex Systems, ECCS'2007*, Dresden, 2007.

Mohammed Rebbah obtained his Engineering diploma in computer science 1993 from University of Sciences and Technology of Oran (USTO) Algeria, and MSC in Computer Science option Pattern Recognition and Artificial Intelligence in 2002. He joined the Computer Sciences at the university of Mascara, Algeria. Currently, he is in final phase of preparing the PhD in computer science. His research interest Grid computing, cloud computing and distributed datamining.

Yahya Slimani studied at the Computer Science Institute of Alger's (Algeria) from 1968 to 1973. He received the B.Sc.(Eng.), Dr Eng and Ph.D degrees from the Computer Science Institute of Alger's (Algeria), University of Lille (French) and University of Oran (Algeria), in 1973, 1986 and 1993, respectively. He is currently Full Professor at the Department of Computer Science of Faculty of Sciences of Tunis. His research activities concern Datamining, parallelism, distributed systems and Grid Computing. Dr. Yahya Slimani has published more than 200 papers from 1986 to 2010. He contributed to Parallel and Distributed Computing Handbook, Mc Graw-Hill, 1996. He is currently Scientific Expert for the European Union. He joined the Editorial Boards of the Information International Journal in 2000, the J.UCS Journal and others journals.

Abdelkader Benyettou is a Professor of electrical engineering at the University of Science and Technoogy of Oran (USTO), Algeria. He received his BSc of engineering in 1982 from the institute of Telecommunications of Oran, and the MSc degree in 1986 from the Universtity of Sciences and Technology of Oran, Algeria. In 1987, he joined the Computer Sciences Research Center of Nancy, France, where he worked until 1991 on Arabic speech recognition by expert systems and received his PhD in electrical engineering in 1993 from the University of Science and Technology of Oran. His interests are in the area of speech recognition, neural networks, and machine learning. He has been the director of the Signal-Speech-Image- SIMPA Laboratory, Department of Computer Science, Faculty of Sciences, University of Sciences and Technology of Oran, since 2002.