Tolérance aux pannes dans les grilles de calcul

Bakhta Meroufel[1], Ghalem Belalem[2] , Nadia Hadi[3]

[1]bmeroufel@yahoo.fr , [2]ghalem1dz@univ-oran.dz, [3]yhana9@yahoo.fr
Département d'Informatique
Faculté des Sciences
Université d'Oran (Es Sénia)

Résumé :

Les grilles de calculs sont des systèmes distribués dans lesquels les pannes existent et ne sont pas des événements rares mais naturels. Comme il s'agit de connecter beaucoup de sites, eux mêmes potentiellement équipés d'une importante quantité de ressources, la notion de pannes est indissociable des environnements de grilles.

Ce présent travail consiste à construire une topologie logique basée sur la clusterisation hiérarchique non couvrante. Sur cette topologie, nous proposons un mécanisme de tolérance aux pannes qui est divisé en deux phases : (i) La première est « Prédiction des pannes » : chaque père surveille ses fils, si un de ses fils susceptible d'être défaillant, le père doit réagir pour protéger les répliques de ce nœud, donc il doit prendre des décisions adéquates en respectant l'espace mémoire et la demande sur cette réplique, (ii) La deuxième phase est la « détection de panne »: chaque nœud envoie périodiquement un message de vie a son père, si ce message n'arrive pas après un certain temps donné, le père considère son fils comme un nœud défaillant et il déclenche l'étape d'auto stabilisation.

Mots clé : Grille de calcul et de données, réplication, arbre recouvrant, tolérance aux pannes, clusterisation (regroupement),

1 Introduction

La perte d'un nœud dans la grille peut avoir des incidences sur le fonctionnement du système de grille et causer des pertes de données (plus souvent des répliques d'un objet donné) qui existent sur ces nœuds.

Dans la littérature [2][10][13][14], il existe plusieurs manières d'envisager une panne dans un système réparti à grande échelle telles que les grilles de calcul (Figure 1), nous pouvons citer :

a) **La réplication (Masking)** : La tolérance aux pannes par duplication consiste en la création de copies multiples des composants sur des processeurs différents. Cette approche par duplication rend possible le traitement des pannes en les masquant.

b) **Forward recovery :** Un point de reprise est un ensemble d'éléments (l'état de la mémoire par exemple) permettant de relancer ce processus en cas de défaillance d'un nœud.
c) **Backward recovery (auto-stabilisation) :** Dijkstra [1] définit un système réparti comme étant *auto-stabilisant* si, indépendamment de son état initial, ce système retourne à un *état légitime* en un nombre fini d'étapes, c'est-à-dire un état à partir duquel le système fonctionne correctement jusqu'à ce qu'une nouvelle défaillance survienne.

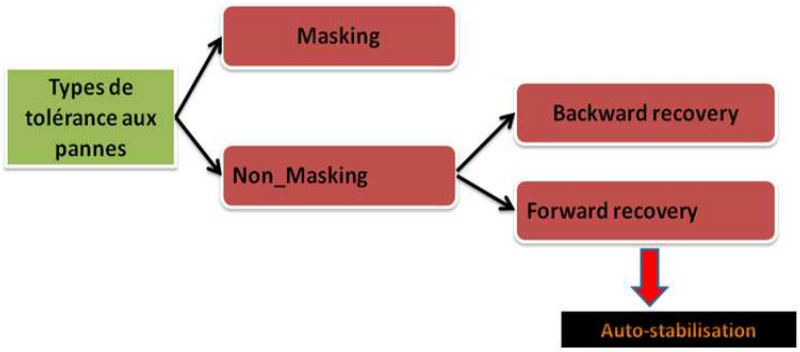

Fig.1. Les mécanismes de tolérance aux pannes

Notre proposition consiste à établir un mécanisme de prédiction des défaillances et de combiner deux techniques de tolérance aux pannes :

- La réplication pour tolérer la panne d'une donnée.
- L'auto stabilisation pour tolérer la topologie après pannes.

Le reste de l'article est organisé comme suit : La section 2 présente l'état de l'art sur les mécanismes de tolérance aux pannes dans les différents intergiciels et les plates formes des grilles de calcul. Dans la section 3, nous donnons une description du système sur lequel notre travail est basé. Nous réservons la section 4 à notre proposition de la tolérance aux pannes. Cette proposition est basée sur trois phases à savoir : la prédiction, la réplication et l'auto stabilisation du système. Nous terminons par une conclusion et des perspectives pour nos travaux futurs.

2 Etat se l'art

Condor [3] utilise les points de reprises pour tolérer les fautes mais il ne supporte pas la perte du coordinateur qui joue le rôle de service d'information et d'ordonnanceur. Il en est globalement de même pour toutes les approches centralisées.

Globus [4] peut survivre à des défaillances de nœuds qui toucheraient son système d'information (MDS) en utilisant le principe de réplication. LDAP est tout à fait adapté à cela.
De plus, si une partie de MDS, non répliquée, est perdue lors d'une défaillance, seulement le sous-arbre associé est perdu, ce qui n'est pas forcément catastrophique si la défaillance n'a pas lieu tout près de la racine de l'arbre. PUNCH [5] utilise aussi le principe de réplication pour assurer la continuité du service malgré les défaillances.
MPICH-V [6] est conçu pour tolérer la volatilité des nœuds mais il repose sur un ensemble de ressources stables pour le stockage des messages notamment.
Des approches complètement distribuées tolèrent beaucoup mieux la perte de nœuds puisque les services ne sont pas centralisés sur un seul nœud. C'est par exemple le cas de Vishwa [7] ou Zorilla [8] qui sont fondés sur des approches pair-à-pair. Toutefois, le système NodeWiz [9] fondé sur une approche pair-à-pair ne supporte pas les défaillances de nœuds.
Dans [15] nous trouvons une table de comparaison entre les différents intergiciels, et parmi les critères de comparaison, nous pouvons noter le mécanisme de suivi du système après pannes, l'existence des mécanismes de détection des pannes et les supports pour la tolérance.
Multi-agent DARX [10] utilise la réplication afin de fiabiliser des agents logiciels. Ce système propose d'adapter le mécanisme de réplication pour tolérer les fautes ainsi que le nombre de copies en fonction : 1) des agents répliqués, et 2) de l'estimation du niveau de risque de fautes.
Un compromis entre les systèmes de MVP et les systèmes pair-à-pair à grande échelle est proposé par la plate-forme JuxMem [11], un service de partage de données modifiables pour la grille. Il permet l'expérimentation des stratégies de tolérance aux fautes par l'utilisation des points de reprise et la réplication des fournisseurs.

3 Description de notre système

Le système proposé est bien adapter aux environnements à grande échelle où il n'existe pas une mémoire partagée. La communication est faite par le passage des messages. Chaque élément du système est identifié par un identificateur unique et des valeurs locales qui indiquent sont degré de vivacité.
Le nœud peut lire et modifier son propre état mais il n'accède à l'état de ces voisins que par lecture. Il peut stocker une ou plusieurs répliques de différents fichiers. La réplique est identifiée par un identificateur (ID) et un numéro de version.

3.1 Topologie logique

Il est difficile de gérer une topologie aléatoire avec des milliers des nœuds et arêtes. Pour cette raison nous trouvons la notion de la topologie logique qui permet de réorganiser les réseaux sous une forme favorable sans toucher cette topologie physiquement.

Dans notre travail, nous utilisons une clusterisation hiérarchique non couvrante (Figure 2). Cette topologie a plusieurs avantages parmi lesquels nous citons :

- Minimisation du temps de réception de chaque message (la gestion et la communication et faite au niveau de chaque cluster).
- Réduction du nombre de messages échangés grâce à l'architecture arbre.

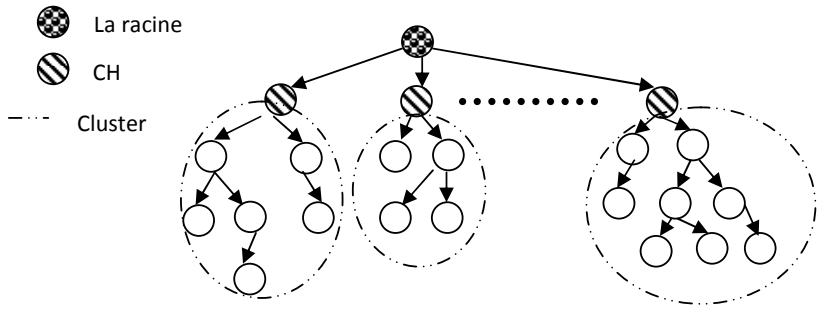

Fig.2. La topologie logique

La racine de la topologie contient la liste des répliques de chaque cluster ce qui permet le bon routage de la requête qui parvient d'un Cluster-Head (CH).
Le Cluster-Head : présente aussi la racine de l'arbre à l'intérieur du cluster et il permet de gérer les nœuds localement et de communiquer avec les autres clusters.
Le nœud représente un élément de stockage qui contient les données et il ne communique qu'avec les nœuds de même cluster.

4 Proposition d'un mécanisme de tolérance aux pannes

Nous décrivons les différentes techniques utilisées pour la tolérance aux pannes. Nous pouvons noter que chaque cluster peut déclencher et exécuter ces techniques de tolérance.

4.1 Prédiction de la panne

Chaque nœud possède des valeurs locales qui indiquent son degré de vivacité (énergie, durée de vie,..). Si cette dernière est inférieure à un certain seuil donné, le nœud informe son père qui doit réagir pour protéger les répliques de son fils:

- Si le père possède les répliques de son fils, il ne doit pas réagir.

- Sinon il envoie une demande à ses voisins (père et fils) pour stocker une ou plusieurs de ses répliques.
- Le nœud qui accepte la demande envoie un OK et la fréquence de demande sur cette réplique sinon il passe la demande à ses voisins.
- Quand le père reçoit ces OK il classe les fréquences de demande de la réplique par ordre croissant puis il envoie la réplique au nœud qui a la plus grande fréquence d'accès.
- Si la réplique est rare (un faible degré de réplication), le père peut envoyer cette réplique à plusieurs demandeurs.
- Les nœuds qui n'acceptent pas de stocker la donnée sont :
    - ❖ Les nœuds qui ont déjà cette réplique ou ils ont des voisins qui possèdent cette réplique.
    - ❖ Les nœuds qui n'ont pas d'espace de stockage suffisant.
    - ❖ Les nœuds qui ne reçoivent aucune demande sur cette réplique (la fréquence de demande sur cette réplique est nulle).
    - ❖ Le cluster-head.

Nous supposons que le temps entre la prédiction de la panne et la panne réelle doit être inférieur au temps de la réaction du père.

4.2 Détection des pannes

Fischer, Lynch et Paterson [12] ont prouvé que dans un système réparti asynchrone, il n'existe pas d'algorithme déterministe qui résolve le problème du consensus lorsqu'un seul processus tombe en panne.
Les travaux de Chandra and Toueg [13] sont les premiers qui ont proposé « *unreliable failure detectors* » où ils ont montré que par un ajout de ce type de détecteur au système asynchrone, il est possible de résoudre le problème de consensus.
Dans notre travail la détection des pannes est faite par l'envoi des messages de vie [2] (ce modèle permet de minimiser le nombre des messages nécessaire pour la détection).
Chaque nœud envoie périodiquement un message de vie à son père, si après certain temps ce message n'arrive pas, le père déclare que son fils est nœud en panne et il déclenche la phase d'auto-stabilisation pour garder la connectivité de la topologie. La panne des Cluster-Head est détectée par la racine. Pour détecter la panne de la racine, elle envoie le message de vie à ses fils.

Nous avons proposé deux types de messages de vie selon l'état de l'émetteur. Un nœud peut avoir un des deux états, soit suspect (le degré de vivacité < un seuil) sinon le nœud est considéré comme correct.
Si le nœud émetteur est correct le message de vie sera de la forme : <MV1, ID émetteur, IN, OUT > où
MV1 : indique un message de vie d'un émetteur correct.
IN : contient la liste des répliques qui ont été ajoutées dans l'émetteur.

OUT : contient la liste des répliques qui ont été supprimées dans l'émetteur (soit par l'utilisateur ou par le système). Généralement IN et OUT contiennent un seul identificateur d'une réplique.

Cette technique permet de contrôler la dynamique des répliques.
Si l'émetteur est suspect, le message de vie sera de la forme :
< MV2, ID émetteur, numéro de séquence >.
MV2 : indique un message de vie d'un émetteur suspect.
Le numéro de séquence indique le nombre des messages envoyés après la prédiction de panne.
Cette valeur permet de détecter la prédiction incorrecte de panne. Ce type de message permet à un père suspect de contrôler un fils suspect.

On peut avoir trois états :

1. Si le père continuera à recevoir les messages de vie d'un fils suspect, alors après certain nombre de ces messages, le père déclare que la prédiction choisie sur l'état de son fils est incorrecte.
2. Si les messages de vie d'un fils suspect n'arrivent pas alors le père déclare que ce nœud est dans un état de panne et que la prédiction est correcte.
3. Si les messages de vie d'un fils correct n'arrivent pas alors il est déclaré en panne malgré qu'il n'était pas suspect et les répliques de ces nœuds seront perdus.

4.3 L'auto-Stabilisation

En cas de panne d'un nœud, la topologie logique sera déconnectée, c'est-à-dire, qu'il y aura des nœuds qui n'ont aucune arête connectée avec la topologie (sauf dans le cas de panne d'une feuille). L'auto stabilisation permet de passer de cet état illégitime à un état légitime où tous les nœuds sont liés (Figure 3).

- Cette phase ne sera déclenchée qu'après une panne.

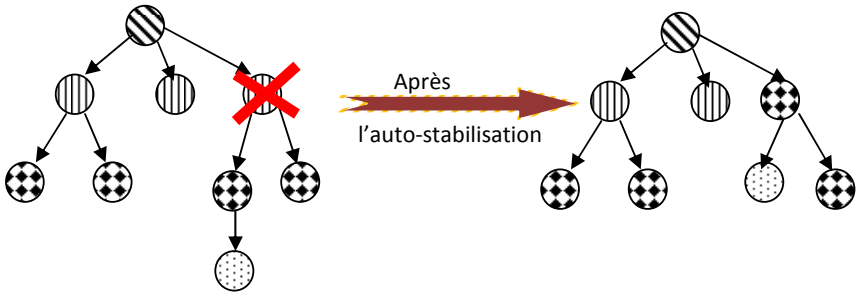

Fig.3. L'auto-Stabilisation après panne

L'auto-stabilisation n'exige pas un état initial et elle est adaptée à la nature dynamique du réseau [14]. Le seul inconvénient de cette technique est qu'elle donne des résultats incorrects pendant la stabilisation du système (phase transitoire du système).

4.4 Gestion de la dynamicité des nœuds

La dynamicité des répliques est gérée par l'envoi des messages de vie qui ont comme émetteur un nœud correct.
A chaque fois que le père reçoit ce message il vérifie les listes IN et OUT pour la mise à jour de sa vue des répliques des fils.
La dynamicité des nœuds correspond au scénario suivant :
Si un nouveau nœud arrive, il envoie une demande d'inscription à tous ces voisins physiques. Les voisins qui acceptent cette inscription, répondent par un message qui contient : son ID, le niveau, charge et la connectivité.
Le nouveau nœud choisit comme père le nœud qui a moins de charges et de connectivité et il commence à lui envoyer les messages de vie.
Si un nœud sort du système, il signale à son père par l'envoie de messages de désistement et le père doit réagir de la même façon que le cas d'une panne.

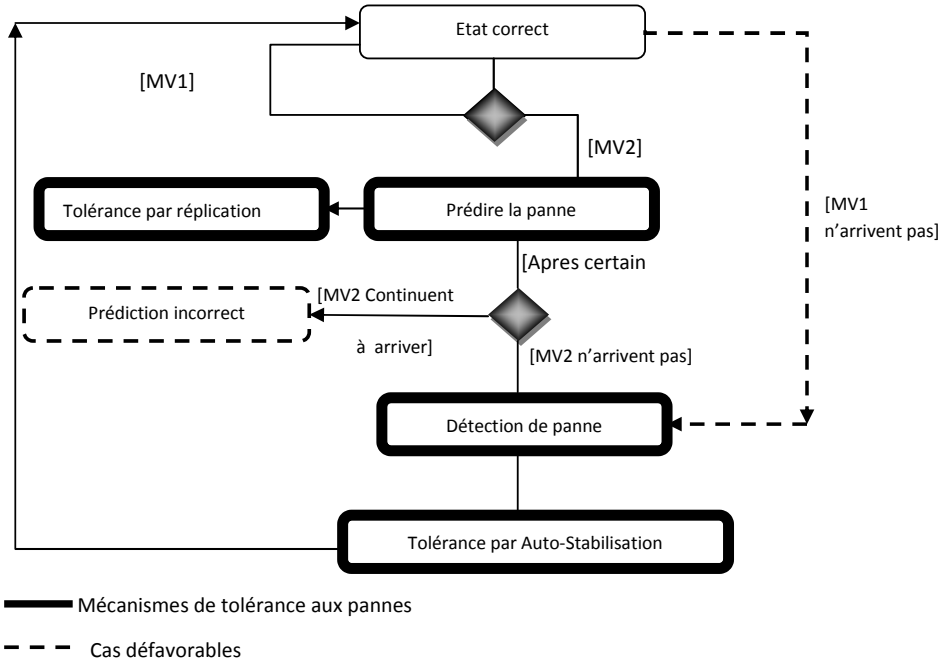

Fig.4. Notre proposition de tolérance aux pannes

5 Conclusion

Dans ce présent rapport, nous avons présenté notre proposition de tolérance aux pannes qui permet de réagir avant que la panne se présente pour protéger les répliques et augmenter la disponibilité des données, et après l'arrivée de la panne pour protéger la connectivité de la topologie.

Notre approche est en cours de réalisation. En perspective, à moyen court, nous pensons à étudier le comportement de notre approche d'un point de vue passage à l'échelle, et de la comparer avec les autres approches classiques. Comme perspective à long terme, nous proposons d'étendre l'approche proposée par un aspect intelligent dans les prises de décision et cela par l'intégration des agents au sein de chaque cluster dans un but d'implémenter une intelligence distribuée coopérative entre les agents.

References

1. E.W. DIJSTRA. Self stabilizing systems in spite of distributed control. Communications of the ACM, 17(11):643–644, 1974.
2. B. HAMID. Distributed Fault-Tolerance Techniques for Local Computations, these de doctorat, Juin 2007
3. M. LITZKOW, M. LIVNY, M. MUTKA. Condor - A Hunter of Idle Workstations . In 8th International Conference of Distributed Computing Systems, June 1988.
4. I. FOSTER ,C. KESSELMAN. Globus: A Metacomputing Infrastructure Toolkit. International Journal of Supercomputer Applications, 11(2):115–128, 1997.
5. N.H. KAPADIA , José A. B. FORTES. PUNCH: An architecture forWeb-enabled wide-area network-computing. Cluster Computing, 2(2):153–164, 1999.
6. A. BOUTEILLER, T. HERAULT, G. KRAWEZIK, P.LEMARINIER, F. CAPPELLO. MPICH-V Project: A Multiprotocol Automatic Fault Tolerant MPI. International Journal of High Performance Computing Applications, 20(3):319– 333, 2006.
7. G. Tarun VENKATESWARA REDDY M., Vijay Srinivas A. Janakiram D. Vishwa: A Reconfigurable Peer-to-Peer Middleware for Grid Computations. In Proceedings of the 35th International Conference on Parallel Processing, pages 381–390, Ohio, USA, August 2006. IEEE Computer Society.
8. N. DROST, R. V. van NIEUWPOORT, H. E. BAL. Simple locality-aware co-allocation in peer-to-peer supercomputing. In Proceedings of the Sixth International Workshop on Global and Peer-2-Peer Computing (GP2P), volume 2, page 14, Singapore, May 2006.

9. S. BASU, S. BANERJEE, P. SHARMA, S.Ju LEE. NodeWiz: peerto- peer resource discovery for grids. In Proceedings of the IEEE International Symposium on Cluster Computing and the Grid 2005 (CCGrid 2005), pages 213–220, Cardiff, UK, May 2005.

10. O. MARIN. The DARX Framework: Adapting Fault Tolerance For Agent Systems. These de doctorat, 'Université de HAVE, December 2003.

11. J.F. Deverge. Cohérence et volatilité dans un service pair-à-pair de partage de données. IRISA, Projet Paris, Juin 2004.

12. M. J. Fischer, N. A. Lynch, M. S. Paterson. Impossibility of distributed consensus with one faulty process. Journal of the ACM, 32(2):374-382,April 1985.

13. T. D. CHANDRA and S. TOUENG. Unreliable failure detectors for reliable distributed systems. Journal of the ACM, 1996.

14. F. REICHENBACH. Service SNMP de détection de faute pour des systèmes répartis. Ecole polytechnique de LAUSANE, Fevrier 2002

15. E. JEANVOINE. Intergiciel pour l'exécution efficace et fiable d'applications distribuées dans des grilles dynamiques de très grande taille. thèse de doctorat, l'Université de Rennes, novembre 2007.