Méthode de réplication basée sur les plans pour la tolérance aux pannes des systèmes multi-agents

Alessandro Luna Almeida, Samir Aknine, Jean-Pierre Briot, Jacques Malenfant

Méthode de réplication basée sur les plans pour la tolérance aux pannes des systèmes multi-agents

Alessandro de Luna Almeida — Samir Aknine — Jean-Pierre Briot — Jacques Malenfant

LIP6, Université de Paris 6
8 rue du Capitaine Scott, 75015 Paris, France
{Alessandro.Luna-Almeida, Samir.Aknine, Jean-Pierre.Briot, Jacques.Malenfant}@lip6.fr

RÉSUMÉ. *L'importance des applications multi-agents et le besoin inhérent d'une meilleure qualité de service pour ces systèmes justifient l'intérêt croissant pour la tolérance aux pannes dans ces systèmes. Dans cet article, nous proposons une méthode originale utilisant la réplication comme technique de fiabilisation pour les systèmes multi-agents. Notre méthode se distingue des autres approches par la construction automatique d'une politique de réplication adaptative. Cette méthode basée sur l'utilisation des plans des agents permet la réplication des agents les plus critiques évitant ainsi leurs pannes. La politique de réplication est déterminée en fonction de la criticité des actions des agents dont les plans représentent à la fois les comportements collectifs et individuels des agents dans leur application. Par ailleurs, les stratégies de réplication appliquées à un moment donné au système multi-agent sont modifiées progressivement par le système de réplication pour mieux gérer la dynamique du système multi-agent.*

ABSTRACT. *The growing importance of multi-agent applications and the need for a higher quality of service in these systems justify the increasing interest in fault-tolerant multi-agent systems. In this article, we propose an original method for providing dependability in multi-agent systems through replication. Our method is different from other works because our research focuses on building an automatic and adaptive replication policy where critical agents are replicated to avoid failures. This policy is determined by taking into account the criticality of the plans of the agents, which contain the collective and individual behaviours of the agents in the application. The set of replication strategies applied at a given moment to an agent is then fine-tuned gradually by the replication system so as to reflect the dynamicity of the multi-agent system.*

MOTS-CLÉS : *Planification, Adaptation, Résistance aux pannes.*

KEYWORDS : *Planning, Adaptation, Fault-tolerance.*

1. Introduction

Les systèmes multi-agents représentent une approche très prometteuse pour la conception d'applications coopératives et réparties (par exemple, la gestion de crise, le contrôle aérien, la gestion de processus, le e-commerce, gestion de réseaux...). Etant des systèmes répartis, les SMA sont aussi sujets aux pannes de diverses origines telles que : les erreurs de programmation, le manque de ressources, les pannes des machines ou du réseau.

Cependant, pour qu'un système puisse poursuivre son fonctionnement en dépit de l'occurrence de pannes, différentes approches de tolérance aux pannes ont été proposées. Ces approches sont basées sur le concept de la redondance : réplication des composants critiques. Dans la plupart des cas, la stratégie de réplication est décidée et appliquée de façon statique, avant même l'exécution de l'application.

Toutefois, les applications récentes, particulièrement, celles utilisant les systèmes multi-agents, peuvent être très dynamiques en raison des processus de réallocation des tâches, d'organisations flexibles, de replanification, des changements de rôles des agents, etc. Ainsi il est très difficile pour le concepteur de décider quels composants (agents) sont les plus critiques, puisque la criticité des agents peut être très variable. La réplication de tous les agents n'est donc pas une approche raisonnable, dans ce cas, vu que les ressources disponibles sont souvent limitées, mais que le coût de cette réplication peut aussi dégrader les performances du système de manière très significative.

En conséquence, il est nécessaire de répliquer de façon automatique et dynamique. Cela implique l'élaboration de mécanismes pour déterminer quand répliquer les agents, quels agents doivent être répliqués, quel est le nombre de réplicats à définir et où déployer ces réplicats.

Dans cet article, nous présentons notre approche de fiabilisation de systèmes multi-agents. Notre méthode est basée sur le concept de criticité, une valeur associée à chaque agent afin de refléter les effets de sa panne sur le système. Cette valeur est calculée en utilisant les plans de l'agent. Un mécanisme de tolérance aux pannes basé sur les plans traite de ce problème de façon préventive. Cette technique est très prometteuse puisqu'elle tient compte des prévisions sur les comportements futurs des agents et de leur influence les uns sur les autres dans la société. Notons que dans le cadre de notre projet, d'autres types d'informations pour calculer la criticité, par exemple, basée sur la notion de rôle ont été étudiées [GUE 05].

Cet article est organisé comme suit : la deuxième section décrit l'architecture du middleware de réplication utilisée et l'architecture de contrôle de réplication. La section 3 explique la méthode de réplication sur la base des plans des agents utilisée comme source de fiabilisation dynamique. La section 4 présente l'état de l'art. Enfin, la dernière section présente des conclusions et les perspectives de recherches futures que nous envisageons.

2. Architecture

2.1 *DARX*

La première étape du projet DARX a consisté en la conception et le développement du framework de réplication dynamique [MAR 01]. DARX se fonde sur la notion de groupe de réplication (GR). Chaque agent de l'application est associé à un GR, que DARX manipule de manière à rendre la réplication transparente pendant l'exécution de l'application. Chaque GR contient exactement un régisseur, qui communique avec les autres agents. La cohérence entre le régisseur et les autres membres du GR est garantie par les stratégies de réplication. Différentes stratégies de réplication (passive, active, …) peuvent être appliquées dans un même GR. Une politique de réplication se définit par l'ensemble des membres formant un GR ainsi que par les stratégies appliquées aux membres de ce groupe.

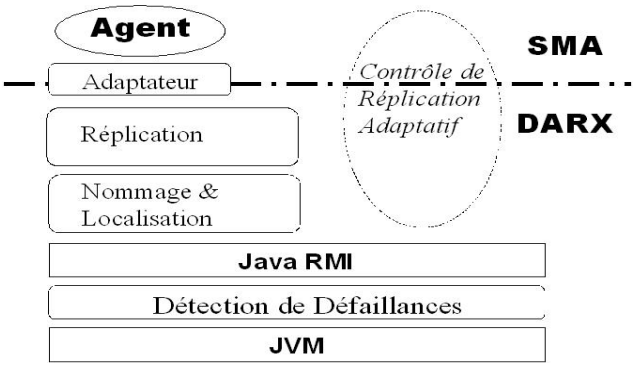

Figure 1. *Les services du framework DARX*

Comme le montre la Figure 1, DARX offre plusieurs services. La détection de défaillances permet de suspecter les pannes de machines et de processus ; sans ce service, le système ne pourra pas vérifier si un agent est encore vivant et qu'il fonctionne dans un environnement qui n'est pas synchrone [FIS 85]. Chaque tâche répliquée possède un identifiant global au sein de l'application. Cet identifiant est défini par le service de nommage, il est ensuite utilisé par le service de localisation pour localiser les agents si nécessaire. Le service de réplication est utilisé par le module de contrôle adaptatif de réplication pour répliquer les agents. DARX peut être facilement intégré à n'importe quelle plateforme d'agent au moyen d'un adaptateur. L'implémentation courante fournit déjà l'intégration aux plateformes DIMA et Madkit.

2.2. *Architecture de Contrôle Adaptatif de Réplication*

Le module de contrôle adaptatif de réplication, représenté dans la Figure 2, est inspiré de l'architecture proposée par [GUE 04]. Nous associons un agent moniteur à chaque agent du système et un moniteur hôte à chaque machine.

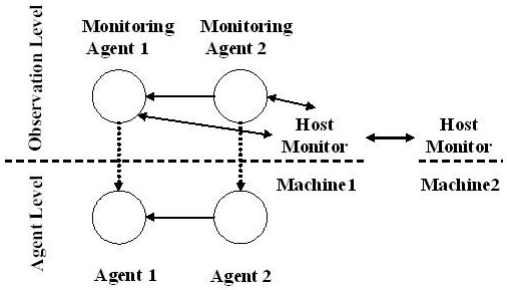

Figure 2. *Architecture de contrôle de réplication*

L'agent moniteur reçoit les plans locaux de l'agent observé. Il est responsable du calcul et de la mise à jour de sa criticité. Comme nous verrons dans la section 4, le calcul de criticité d'un agent peut dépendre de la criticité d'autres agents (en raison des dépendances possibles entre leurs tâches). Ainsi leurs agents moniteurs respectifs doivent s'échanger des informations.

Chaque moniteur hôte contient des informations locales, comme :

– la somme des criticités des agents déployés dans sa machine (les criticités sont obtenues par les agents moniteurs représentés sur la Figure 2) ;

– le nombre total de réplicats dans sa machine et le nombre de réplicats encore disponibles ;

– la fiabilité de sa machine.

Les moniteurs hôtes échangent des informations locales afin d'en déduire des informations globales du système (nombre de réplicats global, somme de criticité de tous les agents, …) et, par conséquent, rendre possible le mécanisme de réplication choisi.

2.3. *Quelles informations utiliser pour calculer la criticité*

Afin de calculer la criticité d'un agent, nous utilisons plusieurs types d'informations. Dans ce projet, nous avons déjà étudié les suivantes :

– Informations du niveau système : charge de communication, temps de traitement [GUE 04].

– Informations du niveau sémantique : le rôle pris par un agent au sein de l'organisation (ex., rôle de broker, manager…) [GUE 05].

Dans cet article, nous proposons d'utiliser d'autres types d'informations du niveau sémantique, les plans des agents ainsi que les dépendances entre leurs tâches.

3. Notre méthode d'évaluation de criticité basée sur les plans

Dans notre modèle, nous considérons que chaque agent du système doit connaître quelles séquences d'actions (plans) ce dernier doit exécuter afin d'atteindre son but courant. Cependant, puisque des événements inattendus peuvent survenir dans les environnements dynamiques, les agents intercalent habituellement la planification et l'exécution. Par conséquent, leurs plans ne sont établis que pour le court terme. Par ailleurs, nous supposons qu'à tout instant donné l'agent exécute au plus une action.

Une action *a* est définie par un n-tuple (I, D, A, R, C, P), où :

– I est l'identificateur de l'action ;

– D est sa durée prévue[1] ;

– A est l'ensemble d'agents qui effectuent conjointement l'action (A peut être unitaire) ;

– R est l'ensemble des ressources nécessaires ;

– C est la criticité absolue de l'action, une valeur fixe et prédéterminée ;

– P est l'ensemble des actions antécédentes, qui doivent être effectuées avant *a*. L'action *a* est une action descendant de P.

Nous représentons le plan d'un agent par un graphe AND/OR acyclique où chaque noeud représente une action. Les noeuds sont reliés par des arêtes AND ou OR. Un noeud *n* qui est relié à *k* autres noeuds (n_1, n_2, ..., n_k) par des arêtes AND représente une action A_n après laquelle **toutes** les actions A_{n1}, A_{n2}, …, A_{nk} seront exécutés. Par contre, si un noeud *n* est connecté à *k* noeuds (n_1, n_2, ..., n_k) par des arêtes du type OR, il suffit qu'**une seule** des actions A_{n1}, A_{n2}, …, A_{nk} soit exécutée après l'exécution de A_n.

Dans l'exemple de la Figure 3, une fois que $Agent_1$ a effectué l'action *A*, les actions *B* **et** *C* doivent aussi être exécutées. Cependant, à partir de l'action *C*, l'exécution d'une des deux actions *D* **ou** *E* suffit pour que $Agent_1$ valide son plan.

1 Nous supposons que la durée d'une action est une valeur approximative normalisée, indépendante de la machine.

Définition 3.1 : Une *action externe a* est une action appartenant au plan d'un agent $Agent_i$ et qui sera exécutée par d'autres agents ($Agent_i \notin a.A$).

Par exemple, considérons l'action *C* appartenant au plan de $Agent_1$ sur la Figure 3. Puisque cette action est effectuée par $Agent_2$, c'est une action externe dans le plan courant de $Agent_1$.

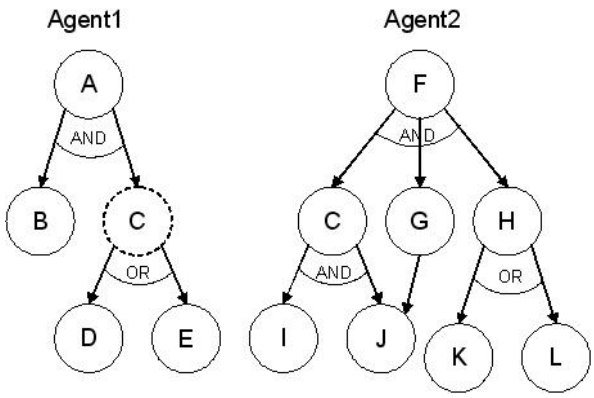

Figure 3. *Exemple de plans de deux agents interagissant*

Définition 3.2 : Une *action terminale* est une action qui n'engendre aucune autre action. Plus formellement :

$$\forall id : Action \cdot (Terminale(id) \Leftrightarrow \neg\exists id' : Action \cdot (id \in id'.P))$$

Dans la Figure 3, *B*, *D*, *E*, *I*, *J*, *K* et *L* sont des actions terminales.

3.1. *Criticité d'une action*

La *criticité absolue (CA)* d'une action est définie sans tenir compte des plans courants des agents. Elle est définie a priori par le concepteur du système et peut être déterminée en fonction d'un certain nombre de paramètres :

– Une action qui peut être effectuée par plusieurs agents peut être considérée comme étant une action peu critique. En effet, si cette action est insérée dans le plan d'un agent, il est plus probable qu'un autre agent l'ait aussi considérée dans son propre plan. En revanche si peu d'agents savent l'exécuter, celle-ci sera moins fréquente dans le plan des autres. Par ailleurs, il est certainement plus facile de la décaler dans le temps si jamais elle échoue.

– Une action dont la durée escomptée est relativement longue peut être considérée plus critique qu'une action plus courte.

– L'ensemble des ressources nécessaires peut également être utilisé comme facteur pour déterminer la criticité absolue. Lorsque beaucoup de ressources sont utilisées par une action, cela peut induire une plus forte criticité que lorsque peu de ressources sont nécessaires.

– Finalement, en fonction du domaine d'application, le concepteur de l'application peut dégager différentes informations sémantiques qui lui permettront de déterminer la criticité d'une action.

La *criticité relative (CR)* d'une action appartenant au plan d'un agent est proportionnelle à la criticité de cet agent lorsque ce dernier l'exécute ou attend qu'un autre agent la lui réalise. La criticité relative d'une action peut être différente pour chaque agent en fonction du plan dans lequel elle figure. Si l'action est externe, sa criticité relative dépend seulement de la criticité relative des actions que l'agent prévoit d'exécuter après cette action.

Si l'agent exécute l'action (possiblement avec d'autres agents), la criticité relative reflète l'importance de cette action dans le système multi-agent. Dans ce cas-ci, elle dépend de la criticité absolue de l'action et de l'utilité de ses résultats à chacun des agents qui dépendent de cette action pour accomplir leurs tâches. En d'autres termes, afin de déterminer la criticité relative d'une action exécutée par un agent, nous devons estimer l'impact de son échec sur d'autres agents.

La criticité relative est calculée comme suit :

– Pour une action externe, elle est égale à la *criticité relative locale (CRL)*. La *CRL* est obtenue en utilisant une fonction d'agrégation AND si l'action est reliée à ses fils par des arêtes AND ou une fonction d'agrégation OR si elle est reliée par des arêtes OR. Les paramètres de ces deux fonctions sont les criticités relatives des fils de cette action. On utilise comme fonction d'agrégation AND la *somme* de ses paramètres et comme fonction d'agrégation OR la *moyenne* des paramètres. Si l'action a seulement un fils, sa *CRL* est égale à la criticité relative de son fils. Si l'action est terminale (i.e., sans aucun fils) sa criticité relative locale est égale à zéro.

– Pour une action non externe *a*, sa criticité relative est égale à sa criticité absolue plus la somme de ses criticités relatives locales dans chacun des plans où elle appartient.

Considérons encore l'exemple de la Figure 3 avec les criticités absolues suivants:

Action	A	B	C	D	E	F	G	H	I	J	K	L
Criticité Absolue	4	8	4	5	1	6	3	2	2	4	3	1

Nous obtenons ainsi les criticités relatives suivantes pour chacune de ces actions:

Action	A	B	C (*Agent*$_1$)	D	E	F	C (*Agent*$_2$)	G	H	I	J	K	L
Criticité Relative	15	8	3	5	1	30	13	7	4	2	4	3	1

Afin d'obtenir ces valeurs, nous utilisons la méthode décrite précédemment. Par exemple, l'action *B* appartenant au plan de *Agent*$_1$ est une action non externe. Sa criticité relative est donc calculée en additionnant sa criticité absolue avec sa criticité relative locale. Puisque cette action est terminale, sa criticité relative locale vaut zéro.

$$CR(B) = CA(B) + CRL(B, Agent_1) = 8 + 0 = 8$$

Cependant, nous calculons la criticité relative de l'action *C* dans le plan de *Agent*$_1$ différemment parce que cette action est externe (elle sera exécuté par l'*Agent*$_2$). Dans ce cas-ci, la criticité relative est simplement égale à la valeur de la criticité relative locale. Afin de calculer la *CRL* de *C* dans le plan de *Agent*$_1$, nous utilisons la fonction d'agrégation (moyenne) avec les criticités relatives des fils de l'action *C* (soit *D* et *E*) comme paramètres.

$$CR(C) = CRL(C, Agent_1) = Moyenne(CR(D), CR(E)) = Moyenne\ (5, 1) = 3$$

Le problème avec cette approche est qu'elle ne considère pas le moment où les actions commenceront réellement à être exécutées. En fait, en utilisant la stratégie décrite ci-dessus, une action terminale qui est située plus loin de la racine du graphe (et ayant probablement une date de début très éloignée) aura un impact identique sur la criticité relative finale de la racine que celui d'une autre action ayant la même criticité absolue mais qui est située à une position plus proche de la racine.

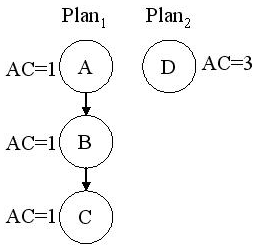

Figure 4. *Impact du temps sur la criticité des actions*

Néanmoins, dans les environnements dynamiques, les actions ayant une date de début plus proche sont plus susceptibles d'être exécutées que des actions ayant une date de début plus éloignée. Sur la Figure 4, l'action *A* du *Plan*$_1$ et l'action *D* du *Plan*$_2$ auront une criticité relative de 3. Cependant, étant donné qu'il n'est pas certain

que les actions *B* et *C* soient exécutées par l'agent, l'action *D* est plus critique dans le $Plan_2$, que l'action *A* du $Plan_1$. En conséquence, nous proposons de multiplier la criticité relative des actions par un facteur qui varie en fonction du temps, en tenant ainsi compte du temps de début prévu pour chaque action. Nous employons la fonction exponentiellement décroissante suivante :

$$RC_{new} = RC_{old} / b^t, \text{ où } b \geq 1 \text{ et } t \text{ est la date de début estimée}$$

Dans l'exemple de la Figure 4, si nous considérons que la durée de toutes les actions est égale à trois unités de temps, et la base *b* d'amortissement est égale à *e*, les criticités relatives obtenues sont les suivantes :

Action	Criticité Relative
A	$CR(A) = (CA(A) + CRL(A, Agent))/e^{ta} = (CA(A) + CR(B))/e^{ta} = (1 + e^{-3} + e^{-9})/e^0 = 1 + e^{-3} + e^{-9}$
B	$CR(B) = (CA(B) + CRL(B, Agent))/e^{tb} = (CA(B) + CR(C))/e^{tb} = (1+ e^{-6})/e^3 = e^{-3} + e^{-9}$
C	$CR(C) = CA(C)/e^{tc} = 1/e^6 = e^{-6}$
D	$CR(D) = CA(D)/e^{td} = 3/e^0 = 3$

Nous calculons les dates de début estimées pour les actions en utilisant une méthode de tri topologique du graphe et en considérant les temps écoulés des actions antécédentes [HIL 80].

3.2. *Criticité d'un agent*

La criticité d'un agent peut être calculée en se basant sur les criticités des actions qui appartiennent à son plan. Un agent qui exécute des actions importantes devrait être considéré comme critique. A un instant donné *t*, la criticité d'un agent est déterminée par la criticité relative de la racine du graphe de son plan courant.

Cependant les systèmes multi-agents sont souvent dynamiques et non-déterministes, il n'est donc pas possible de connaître à l'avance le plan complet de chacun des agents. En fait, pendant l'exécution de leurs plans, un ou plusieurs agents peuvent décider de modifier leurs plans partiels si le contexte d'exécution a relativement changé. Par exemple, cela peut être dû à : un manque de ressources, la dynamicité de la société d'agents (des agents peuvent entrer ou quitter la société), l'impossibilité d'autres agents à accomplir certains de leurs engagements, etc.

En conséquence, nous pouvons considérer que la criticité initiale d'un agent à un instant $t = 0$ est précise, mais qu'elle doit être mise à jour tout au long de l'exécution. La question qui se pose alors est : quand et comment mettre à jour cette

criticité ? Nous proposons deux principales stratégies pour mettre à jour cette criticité : *stratégies temporelles* et *stratégies événementielles.*

Les stratégies *temporelles* sont basées sur des horloges locales associées à chacun des agents. A chaque intervalle de temps Δt, la criticité des agents est réévaluée. L'intervalle de temps peut être fixe ou variable pour gérer la dynamicité du système. Dans ce dernier cas, la longueur de l'intervalle prend une valeur initiale prédéfinie. L'intervalle est réduit si une modification substantielle de la criticité est observée ou, inversement, il sera augmenté si presque aucun changement n'est observé sur la criticité.

Les stratégies *événementielles* sont basées sur les événements critiques qui peuvent changer la criticité de manière significative. Toutes les fois qu'un tel événement est détecté, la criticité est mise à jour. Il y a deux principaux types d'événements : ceux qui dépendent de l'application (exécution d'une action, des changements du plan de l'agent...) et ceux qui sont liés aux pannes (panne d'un agent ou d'une machine).

3.3. *Mécanisme de Réplication d'Agents*

[GUE 04] propose un mécanisme de réplication pour décider quel est l'agent à répliquer et quel est le nombre de réplicats. Dans ce travail, un agent $Agent_i$ est répliqué en fonction de :

– c_i : sa criticité ;

– C : la somme des criticités de tous les agents ;

– *min* : le nombre minimum de réplicats qu'un agent doit avoir (défini par le concepteur) ;

– *max* : le nombre de réplicats disponibles.

Le nombre de réplicats n_i de l'agent $Agent_i$ est déterminé comme suit :

$$n_i = rounded\ (min + (c_i \times max / C))$$

En d'autres termes, il est directement proportionnel au nombre de ressources disponibles et inversement proportionnel à la somme de criticité de tous les agents dans le système. À chaque intervalle de temps Δt, cette valeur est calculée et ensuite utilisée pour mettre à jour le nombre de réplicats de l'agent.

Un problème de ce mécanisme de réplication est qu'il ne prend pas en compte la probabilité de pannes des réplicats. En fait, il est certainement préférable d'avoir une seul réplicat qui aurait une faible probabilité de panne (vu qu'il est déployé sur une machine très robuste) que d'avoir plusieurs réplicats peu fiables. Par ailleurs, dans [GUE 04], ils n'adressent pas le problème du déploiement efficace des réplicats.

Notre mécanisme de réplication considère la probabilité de panne dans les machines. Dans ce nouveau mécanisme, nous définissons la *valeur* d'un réplicat r_k (dénoté par v_k), comme la probabilité pour que sa machine ne tombe pas en panne. Une valeur égale à un sera attribuée à une ressource complètement fiable, tandis qu'une peu fiable aura une valeur proche de zéro. La probabilité de panne d'un ensemble donné de réplicats $R = \{r_1, r_2, \ldots, r_n\}$, est donné par :

$$P\ (Panne\ (R) = 1) = (1\text{-}v_1) \times (1\text{-}v_2) \times \ldots \times (1\text{-}v_n)$$

Soit S la somme des valeurs de tous les réplicats du système, l'$Agent_i$ peut être répliqué avec une valeur totale (t_i) de tous ses réplicats qui est proportionnelle à sa criticité (c_i) et inversement proportionnelle à la somme des criticités de tous les agents (C), comme le montre l'équation suivante :

$$t_i = c_i \times S / C$$

Le système de réplication assigne alors à l'agent l'ensemble des réplicats $R = \{r_1, r_2, \ldots, r_n\}$, avec $v_1 + v_2 + \ldots + v_n \leq t_i$ et dont la probabilité de panne est minimale parmi tous les ensembles de réplicats possibles.

On peut appliquer les mêmes stratégies utilisées pour définir la politique de mise à jour de la criticité d'un agent (*temporel* ou *événementiel*) pour recalculer les valeurs de t_i. Par exemple, on peut définir une fenêtre temporelle variable Δt pour chaque $Agent_i$. Si la valeur totale des réplicats t_i ne change pas de manière significative, la fenêtre Δt peut être augmentée, autrement elle sera diminuée. Une autre possibilité est de recalculer la valeur t_i lorsque la valeur de c_i est mise à jour.

4. Travaux du domaine

Plusieurs approches ont adressé le problème de la tolérance aux fautes. Différents outils fournissent des mécanismes de réplication pour la fiabilisation. Cependant, la plupart d'entres eux ne proposent pas de réplication adaptative.

Hagg [HAG 96] propose d'utiliser des agents sentinelles pour superviser la communication entre agents, pour construire un modèle des autres agents et pour entreprendre des actions de reprise sur erreurs. Les sentinelles analysent toutes les communications dans le système pour détecter les fautes. Cette approche reste trop coûteuse en termes de calculs et de communications. De plus, les sentinelles sont elles-mêmes sources de défaillance.

Decker et al. [DEC 97] décrivent différents niveaux d'adaptation, mais se concentrent sur l'adaptation d'exécution où le clonage d'agent est utilisé comme technique d'équilibrage de charges. Les aspects fondamentaux de la tolérance aux fautes ne sont pas abordés. Kumar et al. [KUM 00] proposent une architecture tolérante aux pannes à base de brokers et le middleware AgentScape offre un service de réplication pour la tolérance aux fautes [BRA 02]. Cependant, dans les

deux cas, la panne d'un agent n'est pas complètement traitée, puisque seulement quelques agents (brokers) ou une partie d'eux peuvent être répliqués.

Fedoruk et al. [FED 02] utilisent aussi la réplication pour la tolérance aux pannes. Leur travail implémente la stratégie de réplication passive d'une manière transparente en utilisant des « proxies ». Tous les messages envoyés et reçus par un groupe de réplication passent pour le « proxy » du groupe.

Kraus et al. [KRA 03] définissent le problème de la tolérance aux fautes comme un problème de déploiement et proposent une approche probabiliste pour déployer les agents dans une application multi-agent. Le problème principal de ces deux travaux est que la réplication est décidée de manière statique avant l'exécution de l'application. Cela n'est pas souhaitable dans le cas des applications multi-agents dynamiques et adaptatives puisque la criticité des agents peut évoluer dynamiquement pendant l'exécution.

D'autres infrastructures logicielles pour la tolérance aux fautes adaptative [CUC 98, FAV 03, KAL 99] existent dans lesquelles des stratégies existantes peuvent être dynamiquement changées. Néanmoins, un tel changement doit être définit au préalable par le concepteur avant l'exécution. Un paramétrage de ces systèmes est également possible mais il reste totalement à la charge du concepteur qui peut l'effectuer de façon interactive avec le système en cours d'exécution.

5. Conclusions et travaux futurs

La recherche sur les systèmes multi-agents a récemment adressé le problème de la fiabilité puisqu'ils doivent souvent fonctionner sans interruption. Pour rendre ces systèmes fiables, nous avons proposé une méthode originale pour évaluer dynamiquement la criticité des agents. Notre approche profite des spécificités d'applications multi-agents et analyse les plans des agents pour déterminer leur criticité. La criticité est alors considérée pour répliquer les agents et maximiser leur fiabilité. Pour valider l'approche proposée, nous utilisons le framework DARX, qui permet de développer notre mécanisme de réplication basée sur les plans.

Une des perspectives de ce travail consiste à comparer notre approche à celles déjà développées. Pour cela, nous réaliserons quelques expérimentations avec deux applications réelles différentes : les assistants personnels de réunions et les agents de patrouille [ALM 04].

Remerciements

Les auteurs tiennent à remercier les membres du projet systèmes multi-agents tolérants aux pannes. Le doctorat du premier auteur est financé par l'organisme CAPES du Brésil.

6. Bibliographie

[ALM 04] Almeida A., Ramalho G., Santana H., Tedesco P., Menezes T., Corruble V., Chevaleyre Y., « Recent Advances on Multi-agent Patrolling », *SBIA 2004*, p. 474-483.

[BRA 02] Brazier F.M.T., van Steen M., Wijngaards N.J.E., « Distributed Shared Agent Representations », In : Marik V., Stepankova O., Krautwurmova H. and Luck M., *Multi-Agent-Systems and Applications II*, Lecture Notes in Computer Science, Vol. 2322, p. 213-220, 2002.

[CUC 98] Cuckuern M. et al., « AQuA : An Adaptive Architecture That Provides Dependable Distributed Objects », *Proceedings of the 17th IEEE Symposium on Reliable Distributed Systems (SRDS'98)*, p. 245-253, West Lafayette, Indiana, October 20-23, 1998.

[DEC 97] Decker K. S., Sycara K., « Intelligent Adaptive Information Agents », *In Journal of Intelligent Information Systems*, vol. 9, p. 239 - 260, 1997.

[FAV 03] Favarim F., Siqueira F., Fraga J.S., « Adaptive Fault-Tolerant CORBA Components », *Middleware Workshops 2003*. p. 144-148.

[FED 02] Fedoruk A., Deters R., « Improving fault-tolerance by replicating agents », *Proceedings AAMAS-02*, Bologna, Italy, p. 737-744.

[FIS 85] Fischer M., Lynch N., Patterson M., « Impossibility of distributed consensus with one faulty process », *JACM*, 32(2) :374-382, 1985.

[GUE 04] Guessoum Z., Briot J.-P., Faci N., Marin O., « Un mécanisme de replication adaptative pour des SMA tolérents aux pannes », *JFSMA*, 2004

[GUE 05] Guessoum Z., Briot J.-P., Faci N., « Towards fault-tolerant massively multiagent systems », In Toru Ishida, Les Gasser and Hideyuki Nakashima editors, *Massively Multi-Agent Systems*, Lecture Notes in Computer Science, Springer Verlag, to appear in 2005.

[HAG 96] Hägg S., « A sentinel Approach to Fault Handling in Multi-Agent Systems », *Proceedings of the Second Australian Workshop on Distributed AI*, Cairns, Australia, August 27, 1996.

[HIL 80] Hillier and Lieberman, *Introduction to Operations Research*, Third Edition, Holden-Day Inc, 1980, p. 246-259.

[KAL 99] Kalbarczyk Z., Bagchi S., Whisnant K., Iyer R.K., « Chameleon : A Software Infrastructure for Adaptive Fault Tolerance », *IEEE Transactions on Parallel and Distributed Systems*, June 1999.

[KRA 03] Kraus S., Subrahmanian V.S., Cihan N., « Probabilistically Survivable MASs », *In Proceedings of Eighteenth International Joint Conference on Artificial Intelligence (IJCAI-03)*.

[KUM 00] Kumar S., Cohen P.R., Levesque, H.J., « The adaptive agent architecture : achieving fault-tolerance using persistent broker teams », *The Fourth International Conference on Multi-Agent Systems (ICMAS 2000)*, Boston, MA, USA, July 7-12, 2000.

[MAR 01] Marin O., Sens P., Briot J.-P., Guessoum Z., « Towards Adaptive Fault-Tolerance for Distributed Multi-Agent Systems », *Proceedings of ERSADS'2001*, Bertinoro, Italy, May 2001.