Un mécanisme de réplication adaptative pour des SMA tolérants aux pannes

Zahia Guessoum [1,2], Jean-Pierre Briot [1], Nora Faci [2] et Olivier Marin [3]

[1] *OASIS/LIP6, 8 rue du Capitaine Scott -F-75015 Paris*

[2] *MODECO/CReSTIC, IUT de Reims, Rue des Crayères, 51000 Reims*

[3] *IIDS, Vrije Universteit, Amsterdam, Pays-Bas*

Zahia.Guessoum@lip6.fr, Jean-Pierre.Briot@lip6.fr, faci@leri.univ-reims.fr, omarin@cs.vu.nl

RÉSUMÉ. *Afin de rendre les systèmes multi-agents à grande échelle fiables, nous proposons une application adaptative de stratégies de réplication. Les agents identifiés comme les plus critiques seront ainsi répliqués pour les protéger contre des possibilités de pannes. Comme la criticité des agents (définie brièvement comme l'impact d'une panne locale à un agent sur un dysfonctionnement de toute l'organisation) peut évoluer pendant le calcul et la résolution de problèmes, il nous faut adapter dynamiquement et automatiquement le nombre de réplicas (copies) des agents, afin de maximiser leur robustesse et leur disponibilité en fonction des ressources disponibles. Nous étudions une approche et des mécanismes pour évaluer la criticité d'un agent donné, basée sur une information sémantique au niveau de l'application (par exemple les dépendances entre agents) et également sur une information statistique au niveau système (par exemple, la charge de communication). Un module de décision choisit à partir de ces informations quelle stratégie appliquer (par exemple, réplication passive ou active) et comment la paramétrer (par exemple, le nombre de réplicats).*

ABSTRACT. *In order to make large-scale multi-agent systems reliable, we propose an adaptive application of replication strategies. Critical agents are replicated to avoid failures. As criticality of agents may evolve during the course of computation and problem solving, we need to dynamically and automatically adapt the number of agent replicas, in order to maximize their reliability and availability based on available resources. We are studying an approach and mechanisms for evaluating the criticality of a given agent (based on application-level semantic information, (e.g., messages intention), and also system-level statistical information, (e.g., communication load) and for deciding what strategy to apply (e.g., active or passive replication) and how to parameterize it (e.g., number of replicas).*

MOTS-CLÉS : *Agent, pannes, réplication, adaptation*

KEYWORDS: *Agents, failures, replication, adaptation*

1. Introduction

Les SMAs offrent une vision décentralisée et coopérative de la résolution de problème. Ils sont donc particulièrement bien adaptés aux problèmes dynamiques et répartis physiquement. Les domaines d'application sont nombreux : gestion de crise, contrôle de processus, ateliers de production flexibles, robotique collective, etc. Cependant, une notion fondamentale des systèmes répartis à large échelle est leur sensibilité aux pannes de diverses origines : machine, réseau, etc.

La tolérance aux pannes est un domaine très étudié dans les systèmes répartis. Les différents mécanismes de réplication proposés ont été appliqués avec succès à plusieurs applications distribuées. Cependant, dans la majorité des cas, la réplication est décidée par le programmeur et elle est appliquée de façon statique avant l'exécution de l'application. Ces différentes applications sont fiables parce que la criticité de leurs composants peut être déterminée par le concepteur et ne varie pas durant l'exécution de l'application. Dans le cas des applications multi-agents dynamiques et adaptatives, la criticité des agents, qui sont souvent adaptatifs, peut évoluer durant l'exécution en fonction de l'évolution des comportements des agents et du comportement collectif. Par ailleurs, les ressources disponibles sont souvent limitées. La réplication simultanée de tous les composants (quelle que soit leur criticité) d'un SMA complexe n'est donc pas toujours possible.

Notre idée consiste à appliquer dynamiquement les mécanismes de réplication en fonction de l'évolution de la criticité des agents. La solution que nous proposons permet ainsi de répondre de façon dynamique aux questions suivantes : 1) quel est l'agent à répliquer ? et 2) quel est le nombre de réplicats ?

Cet article est organisé comme suit : la deuxième section présente l'état de l'art. La troisième section définit la réplication et décrit le framework que nous utilisons pour répliquer nos agents. La quatrième section présente les solutions que nous concevons pour déterminer la criticité des agents de façon automatique et dynamique. Enfin, est brièvement décrite la plate-forme multi-agents que nous proposons pour développer des SMAs tolérants aux pannes. Quelques expérimentations illustreront alors les mécanismes et l'architecture étudiés.

2. Différentes approches pour la tolérance aux pannes

Pour fiabiliser les SMAs, différentes solutions ont été suggérées. Le premier type de solution a été introduit par les SMAs réactifs. Il est fondé sur la redondance. Les systèmes basés sur la métaphore des fourmis offrent un très bon exemple. Les agents réactifs sont souvent similaires, la panne d'un agent n'affecte donc pas le fonctionnement global du système. Cependant, cette redondance ne peut pas être appliquée facilement aux SMAs cognitifs, les différentes copies des agents pouvant provoquer des incohérences. Un contrôle de cohérence de ces copies s'avère en effet nécessaire.

S. Kumar et al. [KUM 00] proposent de faire appel à une équipe d'agents *brokers* pour assurer la tolérance aux pannes des SMAs. Un *Broker* est un agent intermédiaire qui peut offrir divers services tels que la recherche d'agents capables d'effectuer une certaine tâche, router des requêtes et des réponses, etc. Un tel agent peut être intégré dans un SMA dans le but d'assurer une tâche donnée. Toutefois, si cet agent tombe en panne, c'est tout le système qui risque d'en être victime.

S. Hagg [HAG 97] introduit le concept de sentinelle pour protéger les agents des états indésirables. Les sentinelles représentent une structure de contrôle de leur SMA. Elles construisent un modèle de chaque agent et contrôlent les communications pour détecter les pannes. Le concepteur associe à chaque fonctionnalité du SMA une sentinelle. Cette sentinelle contrôle les différents agents qui interagissent pour réaliser cette fonctionnalité. Une analyse de ses croyances sur les autres agents lui permet de détecter les fautes quand elles se produisent. Cette approche semble très intéressante. Cependant, les sentinelles sont également sources de défaillance.

A. Fedoruk et R. Deters [FED 02] proposent une approche de tolérance aux pannes par réplication transparente des agents. Cette approche est basée sur plusieurs idées intéressantes mais elle manque de flexibilité et en particulier de réutilisabilité. Les expérimentations ont été réalisées avec la plate-forme FIPA-OS qui, à notre connaissance, ne fournit aucun mécanisme de réplication. La réplication est simulée par le concepteur avant l'exécution, le mécanisme de contrôle de réplication avancé est donc *ad hoc*.

Dans les systèmes distribués, différents outils intègrent des facilités et des mécanismes de réplication pour construire des applications fiables. Cependant, la majorité des outils n'est pas assez flexible pour implémenter une réplication adaptative. MetaXa [M.G 98] implémente en Java la réplication active et passive de façon flexible. Java est étendu avec une architecture réactive réflexive. Comme dans DARX, la réplication est transparente. Cependant, MetaXa est basé sur un interpréteur JAVA modifié. GARF [GUE 89] réalise des machines Smalltalk tolérantes aux pannes en utilisant la réplication active. Comme MetaXa, GARF recourt à une architecture réflexive et élabore différentes stratégies de réplication. Mais il ne fournit pas de mécanisme adaptatif pour appliquer ces stratégies de façon automatique et dynamique.

3. Réplication

La réplication de données et/ou de calculs est le seul moyen efficace de réaliser la tolérance aux fautes dans les systèmes distribués. Un composant logiciel répliqué est défini comme un composant logiciel qui possède une représentation sur deux ou plusieurs machines.

Il existe deux types fondamentaux de protocoles de réplication : 1) le protocole actif où tous les réplicats effectuent les traitements de façon concurrente, et 2) le protocole passif où un seul réplicat poursuit son exécution et transmet périodiquement son état courant aux autres réplicats pour maintenir la cohérence de l'ensemble du groupe

de réplication. La réplication active entraîne une surcharge importante. En effet, le coût de traitement pour chaque composant est multiplié par son degré de réplication, c'est-à-dire par le nombre de ses réplicats. De même, les communications additionnelles pour maintenir la cohérence au sein du groupe de réplication sont loin d'être négligeables.

Dans le cas de la réplication passive, les réplicats ne sont sollicités qu'en cas de panne. Le principe est le suivant : si le réplicat actif est perdu, un nouveau réplicat est choisi parmi l'ensemble des passifs et l'exécution est redémarrée à partir du dernier état du composant. Cette technique est moins coûteuse que l'approche active, mais le délai nécessaire au recouvrement des traitements perdus est plus important. De plus, il est beaucoup moins évident de garantir un recouvrement total dans l'approche passive, puisqu'on repart forcément du dernier point de mise à jour.

Plusieurs outils [REN 96] intègrent des facilités de réplication pour construire des applications tolérantes aux fautes. Cependant, la majorité des produits n'est pas assez flexible pour implémenter des mécanismes adaptatifs. Rares sont les systèmes qui permettent de modifier la stratégie de réplication choisie durant l'application. Plus encore, aucun système ne donne la possibilité de laisser à l'application répartie le choix de sa politique de réplication.

Le paragraphe suivant présente le framework DARX. Ce dernier fournit un mécanisme flexible de tolérance aux fautes pour les SMAs.

3.1. *DARX*

DARX [MAR 03] est un framework pour concevoir des applications distribuées tolérantes aux fautes. Le modèle de fautes utilisé couvre les pannes franches, c'est-à-dire les défaillances définitives de machines ou de connexions réseau, et dans une certaine mesure les partitionnements réseau. Les fautes byzantines, par exemple celles résultant d'attaques malicieuses, pourraient également être couvertes mais elles ne sont pas encore intégrées au modèle.

Chaque tâche peut être répliquée indépendamment des autres tâches, constituant ainsi un groupe de réplication (GR) qui sert à représenter de manière transparente la tâche concernée auprès du reste de l'application. Le nombre de réplicats et la stratégie interne de réplication d'une tâche spécifique sont ainsi totalement cachés aux autres tâches de l'application. DARX possède plusieurs mécanismes pour contrôler les GRs, dont l'ajout/suppression de réplicats. Il fournit également un multi-cast atomique pour la communication interne aux GRs. Chaque GR contient un unique régisseur qui représente le groupe auprès des autres tâches. En cas de panne du régisseur, un remplaçant est automatiquement élu parmi l'ensemble des réplicats restants. DARX fournit un système de nommage. Chaque tâche répliquée possède un identifiant global au sein de l'application ; cet identifiant est indépendant des caractéristiques du GR ainsi nommé. Afin de tirer profit des capacités de tolérance aux fautes offertes par DARX, chaque agent est enveloppé dans un objet TaskShell (voir figure 1), permettant au système

de manipuler les mécanismes d'exécution et de communication de l'agent. Il devient ainsi possible pour DARX d'agir en tant qu'intermédiaire de l'agent et de décider quand un agent doit être activé ou suspendu, et quand une réception de message doit être effectuée.

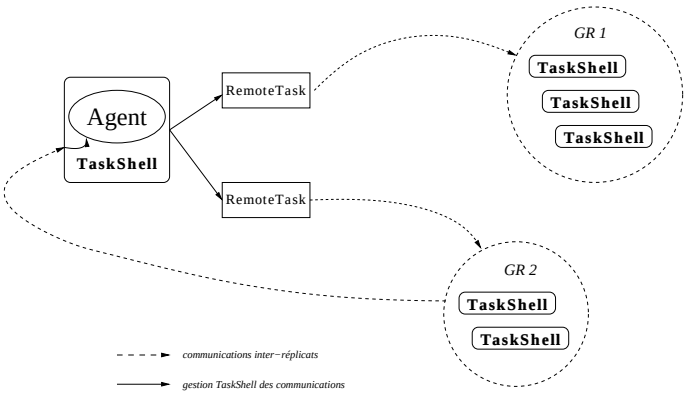

Figure 1. *Gestion de groupes de réplication dans DARX*

Le TaskShell sert également à gérer le groupe de réplication de manière totalement transparente pour les agents ; cela inclut la prise en charge des communications et la préservation de la cohérence dans le groupe de réplication. Le TaskShell du régisseur d'un GR va donc transmettre périodiquement l'état de l'agent qu'il enveloppe aux réplicats passifs, et faire passer systématiquement tous les messages reçus aux réplicats actifs. Les communications, supposées asynchrones dans le modèle, sont interceptées par le TaskShell et gérées au moyen d'un cache de réception. Il est alors possible à la fois de journaliser les messages échangés entre agents pour garantir la cohérence lors de recouvrements, et de garantir le séquencement des messages au sein du groupe ainsi que l'unicité des messages émis vers l'extérieur. La communication entre tâches distinctes se fait au moyen d'un proxy : le RemoteTask. Chaque RemoteTask pointe sur le régisseur courant du groupe de réplication référencé. Ceci permet de communiquer indifféremment avec des entités locales ou distantes, que ce soit des GRs ou des agents.

3.2. *Discussion*

DARX fournit des mécanismes pour répliquer les agents et modifier la stratégie de réplication dynamiquement. Cependant, nous ne pouvons pas toujours répliquer tous les agents d'un système à large échelle parce que les coûts associés deviendraient prohibitifs, à la fois au niveau des temps d'exécution et des ressources exploitées. De plus, le nombre de ressources disponibles varie durant l'exécution du système.

Un bon mécanisme de réplication devrait donc adapter la stratégie de réplication à l'évolution de l'environnement. La solution que nous proposons consiste ainsi à appliquer automatiquement et dynamiquement les mécanismes de réplication aux agents. Cette solution, décrite dans la section suivante, est basée sur un processus d'observation et un processus de monitoring.

4. Criticité d'agents

L'analyse de la criticité d'un agent permet de définir son importance et l'influence de sa défaillance sur le comportement normal et la fiabilité du SMA. Nous nous sommes basés sur les concepts de base des SMAs pour définir cette criticité. L'objectif est d'élaborer une approche automatique afin de ne pas rendre le développement des SMAs plus complexe. Cependant, notre solution peut également utiliser des données fournies *a priori* par le concepteur de l'application. Ce dernier peut, par exemple, choisir de figer la stratégie de réplication d'un agent ou de l'ensemble des agents du système.

Pour nous, la criticité d'un agent dépend de deux types d'informations :

– *Informations du niveau système*. Elles sont basées sur des mesures standard (charge de communication, temps de traitement...). Elles sont notamment utilisées pour évaluer le degré d'activité d'un agent.

– *Informations du niveau sémantique*. Elles dépendent du domaine d'application et du paradigme choisi (e.g. agent).

La résolution de problèmes dans un SMA émerge des interactions entre un ensemble d'entités organisées appelées agents. Un SMA est ainsi principalement caractérisé par les structures organisationnelles qui peuvent être statiques ou dynamiques. Par exemple, Ferber [FER 98] utilise le concept de *rôle* pour définir les structures organisationnelles. Ce concept est bien approprié aux domaines d'applications où les structures organisationnelles sont connues *a priori*. Par ailleurs, dans beaucoup de domaines complexes, les structures organisationnelles et leurs évolutions ne peuvent pas être définies *a priori*. Elles émergent des interactions entre agents. Castelfranchi [CAS 98] cite plusieurs exemples de structures organisationnelles émergentes tels que les réseaux de communication et les réseaux d'interdépendance, etc.

Nous proposons une solution basée sur les *réseaux d'interdépendances*. Notre approche est générique, elle ne dépend pas d'un langage d'interaction ou d'un domaine d'application. Cependant, les agents, qui peuvent être cognitifs ou réactifs, communiquent avec un langage de communication tels que FIPA ACL et KQML.

Les sections suivantes présentent l'architecture proposée et montrent comment les réseaux d'interdépendances sont définis, mis à jour et utilisés pour évaluer la criticité d'agents.

4.1. *Une architecture multi-agents*

L'évaluation de la criticité d'agents est basée sur l'observation. Le SMA est enrichi d'un mécanisme d'auto-observation qui facilite l'adaptabilité dynamique de la réplication. Ce mécanisme permet :

– d'analyser les informations observées afin de déterminer la criticité d'agents,

– d'adapter dynamiquement la réplication.

Dans la majorité des travaux sur l'observation des SMAs, le mécanisme d'observation est centralisé. L'observation est utilisée pour collecter des informations sur l'exécution du système et les analyser à la fin de l'exécution. Le but de cette observation est d'expliquer le comportement du système. Par ailleurs, les domaines d'application considérés ne sont pas très complexes (petit nombre d'agents, structures organisationnelles connues, ...). Ces mécanismes d'observation centralisés ne sont pas bien appropriés aux systèmes à large échelle où les informations observées doivent être analysées en temps réel pour adapter le SMA à l'évolution de son environnement.

Nous proposons de distribuer le mécanisme d'auto-observation afin d'améliorer l'efficacité du système d'auto-observation et sa robustesse. Cette distribution est fondée sur une organisation d'agents réactifs appelés Moniteurs dont le but consiste à observer et contrôler les agents du domaine. Dans notre architecture multi-agents, nous associons un agent Moniteur à chaque agent du domaine (voir figure 2) et un agent Observateur à chaque machine.

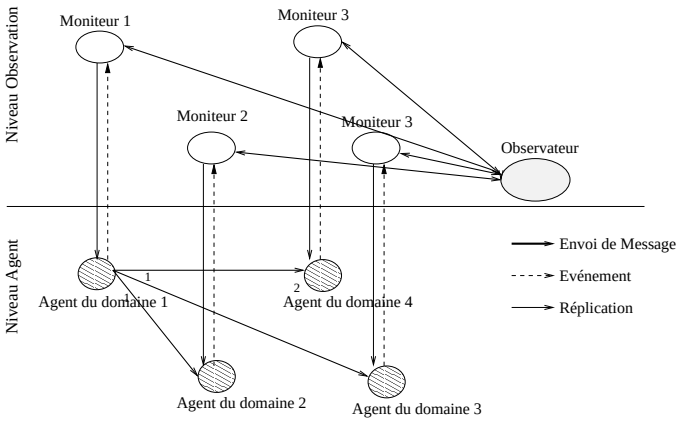

Figure 2. *Architecture multi-agents*

L'observation des agents est basée sur le module d'observation intégré aux serveurs DARX présents sur chaque machine hôte. Les informations fournies par ce module sont gérées par l'agent Observateur. Chaque agent Moniteur s'abonne à cet agent Observateur pour recevoir les informations et les événements correspondant à

l'agent du domaine auquel il est lié. Il analyse ensuite ces informations pour évaluer la criticité de l'agent du domaine et mettre à jour sa réplication.

Les agents de contrôle (Moniteurs et Observateurs) sont organisés de façon hiérarchique. Chaque agent Moniteur ne communique qu'avec l'agent Observateur. Les agents Observateurs s'échangent des informations locales afin d'en déduire des informations globales du système (nombre de réplicats global, quantité d'informations globale échangée entre agents...).

4.2. *Réseaux d'interdépendances*

Dans un SMA, un agent est défini comme une entité autonome. Cependant, les agents ne sont pas complètement autonomes. Pour réaliser leur(s) but(s), les agents n'ont pas toujours toutes les compétences et/ou les ressources nécessaires. Ils dépendent donc des agents qui ont les compétences et/ou les ressources requises pour réaliser leur(s) but(s). Les réseaux d'interdépendances [SIC 94] ont en effet été introduits pour décrire les structures organisationnelles de ces agents. Ils ont été utilisés dans de nombreux travaux qui définissent souvent les réseaux d'interdépendances de façon statique avant l'exécution du SMA. Cependant, les SMAs complexes sont caractérisés par des structures émergentes.

4.2.1. *Définition du réseau d'interdépendances*

A chaque agent du domaine est associé un nœud. Ce dernier définit le rôle organisationnel de l'agent qui montre que celui-ci n'est pas une entité indépendante, il est contraint par (et contraint) son organisation. Ce nœud représente donc l'interface entre l'agent et son organisation.

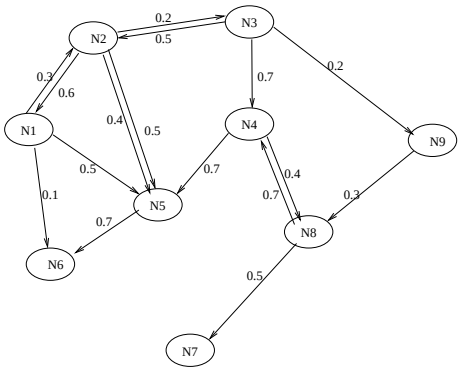

Figure 3. *Exemple de graphe d'interdépendances*

L'ensemble des nœuds, appelé réseau d'interdépendances, est représenté par un graphe orienté (N, L, W). N définit l'ensemble des nœuds du graphe, L les arcs et

W les poids des arcx. $L_{i,j}$ est le lien entre les nœuds N_i et N_j. Ce lien est caractérisé par un poids $W_{i,j}$ qui quantifie l'interdépendance entre les agents associés : $Agent_i$ et $Agent_j$. La figure 3 donne un exemple de graphe où les liens avec un poids nul ne sont pas représentés. Un nœud est ainsi relié à un ensemble de nœuds qui peut être vide (l'agent ne dépend d'aucun autre agent) ou regrouper tous les autres nœuds du réseau. Cet ensemble est dynamique puisque des liens peuvent disparaître et de nouveaux liens peuvent apparaître.

Chaque nœud est géré par l'agent Moniteur associé à l'agent du domaine correspondant. Les nœuds sont automatiquement générés à l'activation du système. L'ensemble initial des liens de chaque nœud est vide.

Pour définir les dépendances $W_{i,j}$ de $Agent_i$ et $Agent_j$ ($Agent_j$ dépend de $Agent_i$), nous proposons de considérer deux types d'informations :

– $QI_{i,j}$: la charge de communication, c'est-à-dire la quantité d'informations envoyées par $Agent_i$ à $Agent_j$,

– $NbM_{i,j}$: le nombre de messages envoyés de $Agent_i$ à $Agent_j$.

Nous proposons de définir $W_{i,j}$ en fonction de $QI_{i,j}$, $NbM_{i,j}$ ainsi que d'informations globales telles que la charge moyenne de communication et le nombre de messages moyen.

4.2.2. *Adaptation de l'interdépendance*

Toute modification de l'agent du domaine entraîne une modification de son nœud. Celle-ci peut ensuite être propagée aux nœuds voisins en modifiant les liens entre le nœud d'origine et les nœuds voisins.

Les modifications du réseau d'interdépendances sont contrôlées par les agents Moniteurs. Ces derniers permettent d'adapter le graphe d'interdépendances aux différents changements des agents du domaine (par exemple, l'arrivée de nouveaux agents...). La mise à jour du graphe est distribuée. Chaque agent met à jour la dépendance de son nœud avec les autres nœuds et envoie les informations concernant son nœud à l'agent Observateur. Les différents agents Observateurs construisent ensuite des informations globales (charge de communication moyenne QI, nombre de message moyen NbM...) et les envoient aux agents Moniteurs.

Pour un intervalle de temps Δt, la charge de communication globale $QI(\Delta t)$ et le nombre global de messages $NbM(\Delta t)$ sont calculés comme suit :

$$QI(\Delta t) = Mop1(QI_{1,1}(\Delta t), QI_{1,2}(\Delta t)..., QI_{n,n}(\Delta t)) \quad (1)$$

$$NbM(\Delta t) = Mop2(NbM_{1,1}(\Delta t), NbM_{1,2}(\Delta t)..., NbM_{n,n}(\Delta t)) \quad (2)$$

où Mop1 et Mop2 sont des opérateurs d'agrégation. Pour nos expérimentations nous avons utilisé la moyenne.

Algorithme 1 : Adaptation des interdépendances d'un agent $Agent_i$. $QI(\Delta t)$ et $NbM(\Delta t)$ sont fournis par l'agent Observateur.
Début
Pour tout j différent de i :

$$myQI = (QI_{i,j}(\Delta t) - QI(\Delta t))/QI(\Delta t) \quad (3)$$

$$MyNbM = (NbM_{i,j}(\Delta t) - NbM(\Delta t))/NbM(\Delta t) \quad (4)$$

Mettre à jour les poids en utilisant la règle suivante :

$$W_{i,j}(t + \Delta t) = W_{i,j}(t) + (a * MyQI + b * MyNbM)/(a + b) \quad (5)$$

Fin
L'algorithme 1 donne un aperçu de l'opération d'adaptation du réseau d'interdépendances. Cette adaptation est fondée sur des informations locales (charge de communication...) et des informations globales (agrégation des différentes charges de communication). Ces informations globales ne sont pas exactes, mais les valeurs approchées sont suffisantes pour notre algorithme. Cet algorithme est utilisé par chaque agent Moniteur pour gérer le nœud associé. Dans cet algorithme, nous n'avons considéré que deux types d'informations : nombre de messages et la quantité d'informations échangés. Ces deux types d'informations sont souvent suffisantes pour évaluer les interdépendances. Cependant, dans beaucoup d'applications, il est nécessaire de tenir compte des :

– types de messages (performatifs par exemple),

– priorité des messages,

– séquences de messages (protocoles utilisés).

L'algorithme proposé pourra en effet être étendu pour intégrer ces différents types d'informations. Des nouveaux algorithmes sont ainsi en cours d'élaboration.

4.3. *Evaluation de la criticité d'agents*

Dans un SMA, l'activité interne d'un agent ne peut pas toujours être facilement observée. L'observation est en effet souvent restreinte aux événements et aux informations du niveau système. Pour évaluer le degré d'activité d'un agent du domaine, nous nous basons sur des mesures du niveau système fournies par le module d'observation de DARX. Pour un agent $Agent_i$ et un intervalle de temps Δt, ces mesures donnent, par exemple, le temps CPU utilisé (cp_i). Ce dernier est utilisé pour évaluer le degré d'activité de l'agent aw_i comme suit :

$$aw_i = cp_i/\Delta t \quad (6)$$

Les interdépendances et le degré d'activité de chaque agent du domaine sont utilisés pour déterminer sa criticité. La criticité de $Agent_i$ peut être calculée comme suit :

$$w_i = (a_1 * AgOp(W_{j,i\,j=1,m}) + a_2 * aw_i)/(a1 + a2) \quad (7)$$

où AgOp est un opérateur d'agrégation, a_1 et a_2 sont les poids donnés aux deux types de paramètres (interdépendances et degré d'activité). Leurs valeurs par défaut sont (a_1=0.5 et a_2=0.5), mais ils peuvent être initialisés par le concepteur.

Chaque agent $Agent_i$ est répliqué en fonction de :

– w_i : sa criticité,

– W : la somme des criticités de tous les agents, W n'est pas calculée à chaque pas,

– rm : le nombre minimum de réplicats, il est défini par le concepteur,

– Rm : les ressources disponibles qui définissent le nombre de réplicats possibles.

Le nombre de réplicats nb_i de $Agent_i$ peut-être déterminé comme suit :

$$nb_i = rounded(rm + w_i * Rm/W) \quad (8)$$

nb_i est ensuite utilisé pour mettre à jour le nombre de réplicats de $Agent_i$.

5. Vers un environnement de développement de SMA résistants aux pannes

Pour faciliter l'intégration des plates-formes multi-agents existantes et le middleware DARX, nous avons élaboré un adaptateur qui permet de réaliser une passerelle entre les plates-formes multi-agents et DARX. Cet adaptateur permet d'encapsuler les agents dans des DarxTask. Les agents peuvent ainsi être distribués et répliqués dynamiquement. Pour la validation de ce travail nous avons utilisé la plate-forme DIMA.

L'architecture modulaire de DIMA a facilité cette intégration. Toute application écrite avec DIMA peut en effet être exécutée avec DIMA+DARX. Dans DIMA, un agent est décrit indépendamment de la plate-forme d'exécution (avec ou sans DARX). Mais au moment de son activation, on choisit la plate-forme d'exécution et la méthode d'activation correspondante. La réception des messages est assurée par la DarxTask, qui lorsqu'elle reçoit un message, le dépose dans la boîte aux lettres de l'agent correspondant. Les agents DIMA héritent ainsi de toutes les caractéristiques des DarxTask. Ils peuvent être répliqués et déployés dynamiquement.

L'exemple test de notre plate-forme est un SMA utilisé en tant qu'assistant pour la prise de rendez-vous automatique. Chaque participant utilisateur possède un agenda électronique qui lui présente son emploi du temps.

5.1. *Evaluation de performances*

Pour valider la plate-forme réalisée (DIMA + DARX), nous avons effectué plusieurs expérimentations afin d'évaluer différents paramètres tels que le temps de transmission de messages et le temps de réplication des agents.

Le premier test s'est porté sur l'évaluation du coût de la transmission d'un message (de taille variable) entre deux agents communicants, activés sur des machines

distantes. Ce test nous permet d'évaluer l'influence de la distribution des agents sur des machines distantes, sachant que le temps de transmission de messages entre agents locaux est négligeable. Nous avons considéré les paramètres suivants :

– taille du message transmis (variations entre 100 octets et 6400 octets),

– nombre de réplicats de l'agent récepteur (en stratégie active) : cas sans réplication, un réplicat, deux réplicats ou trois réplicats.

Note : Les tests que nous avons réalisés, ont été effectués sur six machines de type Intel(R) Pentium(R) 4 CPU 2.00GHz, 526 M Octets de mémoire vive RAM. Sur la

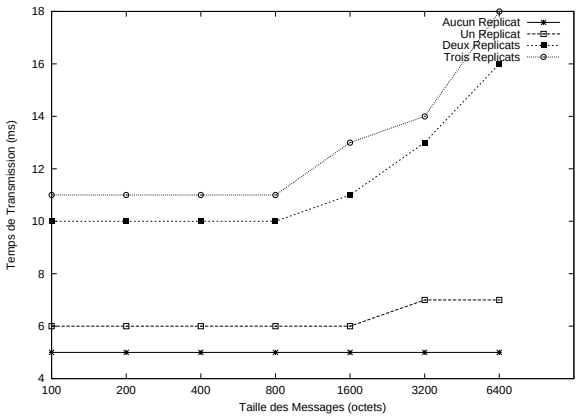

Figure 4. *Temps de transmission de messages*

figure 4, nous remarquons que les temps de transmission de messages entre les deux agents, sans réplication du récepteur, est constant (5ms). La variation de la taille du message entre 100 et 6400 octets n'affecte pas le temps de réponse du système. Pour un seul réplicat actif, nous remarquons que le temps commence à changer à partir d'une taille de message supérieure à 1600 octets. En revanche, pour deux et trois réplicats, la différence des temps commence à être relativement significative à partir d'une taille de messages supérieure à 800 octets. Toutefois, dans le cadre d'applications de SMA, de telles tailles de messages se présentent rarement.

5.2. *Tests de robustesse*

Pour faire les tests de robustesse, nous avons implémenté un simulateur de pannes. Ce simulateur choisit un agent de façon aléatoire et détruit la thread correspondante. Pour chaque expérimentation, nous définissons la durée de simulation (DS), et le nombre de pannes (NP). Le simulateur génère en effet une panne tous les Δt :

$$\Delta t = \frac{DS}{NP} \tag{9}$$

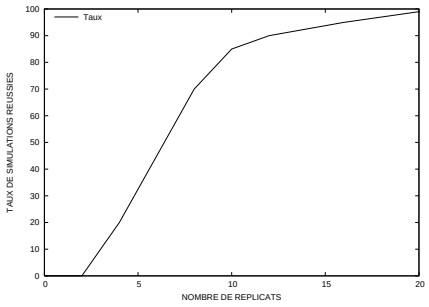

Figure 5. *Taux de simulations réussies pour chaque taux de réplication*

Dans cette expérimentation, nous avons considéré 100 agents distribués sur 10 machines, la durée de simulation *DS* = 10 mn et le nombre de pannes *NP*=100. Nous avons répété l'expérimentation plusieurs fois en variant le nombre de ressources (nombre total de réplicats). La figure 5 montre le taux de succès SR en fonction du nombre de réplicats.

$$SR = \frac{NSS}{TNS} \tag{10}$$

où NSS est le nombre de simulations qui n'ont pas échoué, et TNS est le nombre total de simulations.

Une simulation échoue si une panne détruit un agent critique qui n'a aucun réplicat. Ces expérimentations montrent que le nombre de ressources doit être supérieur au nombre d'agents critiques.

Ces résultats sont préliminaires, mais ils sont encourageants et montrent l'intérêt de l'approche que nous avons adoptée.

6. Conclusion

L'approche de tolérance aux pannes que nous proposons est fondée sur le framework DARX qui offre plusieurs stratégies de réplication et permet de changer dynamiquement la stratégie de réplication. Cette approche présente certaines similitudes avec celle de Fedoruk et Deters [FED 02], du point de vue des mécanismes de tolérance aux pannes (réplication et groupe de communication) introduits. Cependant, plusieurs avantages sont à signaler au niveau architectural et fonctionnel, notamment la gestion des réplicats, et les communications agent-agent et agent-réplicat. Notre approche se veut être plus transparente et complètement distribuée. Nous introduisons le concept d'adaptation dynamique qui, jusqu'à présent n'a jamais été traité dans le cadre d'approches de réplication d'agents.

L'implémentation des mécanismes et des modèles élaborés a permis de développer une plate-forme multi-agents. Cette plate-forme de développement de SMAs intègre les mécanismes de tolérance aux pannes, et offre ce service d'une manière transparente, n'imposant aucune modification fonctionnelle ou structurelle au niveau des comportements des agents développés. Une telle solution devrait pouvoir fiabiliser les applications multi-agents sans aucune contrainte de compétences et de connaissances des agents. Cependant, les expérimentations réalisées sont très simples. Une des perspectives de ce travail consiste en effet à valider cette plate-forme sur une application réelle complexe (large échelle, agents adaptatifs...).

Remerciement

Ce travail a été fait dans le cadre du projet LIP6 ARP *(Agents Résistants aux Pannes)*. Les auteurs tiennent à remercier tous les membres de ce projet.

7. Bibliographie

[CAS 98] CASTELFRANCHI C., « Modelling Social Action for AI agents », *Artificial Intelligence*, vol. 103, 1998, p. 157-182.

[FED 02] FEDORUK A., DETERS R., « Improving Fault-Tolerance by Replicating Agents », *AAMAS2002*, Bologna, Italy, 2002, p. 737-744.

[FER 98] FERBER J., GUTKNECHT O., « Alaadin : a meta-model for the analysis and design of organizations in mutli-agent systems », DEMAZEAU Y., Ed., *ICMAS'98*, Paris, 1998, p. 128-135.

[GUE 89] GUERRAOUI R., GARBINATO B., MAZOUNI K., « Lessons from Designing and Implementing GARF », *Proceedings Objects Oriented Parallel and Distributed Computatio*, vol. LNCS 791, Nottingham, 1989, p. 238-256.

[HAG 97] HAGG S., « A sentinel Approach to Fault Handling in Multi-Agent Systems », ZHANG C., LUKOSE D., Eds., *Multi-Agent Systems, Methodologies and Applications*, n° 1286 LNCS, Springer Verlag, 1997, p. 190-195.

[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* $ICMAS$, Boston, USA, 2000, p. 159-166.

[MAR 03] MARIN O., BERTIER M., SENS P., « DARX - A Framework for the Fault-Tolerant Support of Agent Software », *14th International Symposium on Software Reliability Engineering (ISSRE'2003)*, Denver, Colorado, USA, 2003, IEEE, p. 406-418.

[M.G 98] M.GOLM, « MetaXa and the Future of Reflection », *OOPSLA -Workshop on Reflective Programming in C++ and Java*, Springer Verlag, 1998, p. 238–256.

[REN 96] VAN RENESSE R., BIRMAN K., MAFFEIS S., « Horus : A flexible group communication system », *Communications of the ACM*, vol. 39, n° 4, 1996, p. 76-83.

[SIC 94] SICHMAN J. S., CONTE R., DEMAZEAU Y., « A Social Reasoning Mechanism Based On Dependence Networks », *Proceedings of ECAI'94 - European Conference on Artificial Intelligence*, Amsterdam, The Netherlands, August 1994, p. 188-192.