Réplication dynamique d'une base de données dans un système P2P : Algorithme de gestion efficace des ressources.

Aissatou Diaby Gassama* - Idrissa Sarr**

Département d'informatique
Université Cheikh Anta Diop
Dakar
Sénégal

* Gassama_diaby@hotmail.fr

** idrissa.sarr@ucad.edu.sn

RÉSUMÉ. L'utilisation des réseaux Pair à Pair pour assurer la réplication d'une base de données est une des solutions très utilisées pour éviter les problèmes de congestion induits par les architectures client/serveur et favoriser un meilleur passage à l'échelle. Avec la réplication, les nœuds qui stockent les données (ou répliques) sont généralement structurés en maître esclave afin de pouvoir contrôler la cohérence des données. Les nœuds maîtres sont utilisés pour traiter les mises à jour alors que les esclaves servent à traiter des requêtes de lecture seule. Ceci accroît le débit transactionnel et diminue le temps de réponse. Toutefois, les nœuds d'un réseau P2P sont sujets à des déconnexions intempestives, ce qui impacte les performances globales. En effet, la déconnexion d'un nœud maître peut compromettre la cohérence et l'équilibrage des charges. Pour faire face à ce problème, nous proposons un modèle de gestion dynamique des nœuds maîtres. Notre approche se base sur l'estimation de la durée de vie restante (temps de connexion restant) d'un nœud maître pour le remplacer par un esclave afin d'assurer la réception et l'exécution d'une requête de mise à jour. Nous avons simulé notre solution avec l'outil PeerSim et les résultats obtenus montrent la faisabilité de notre approche et son impact positif sur les performances globales du système.

MOTS-CLÉS : Base de données répliquées, système P2P, tolérance aux pannes et transactions.

1. Introduction

Les systèmes pair à pair, sont utilisés pour partager des données entre utilisateurs se situant dans des sites géographiquement éloignés. Les données partagées sont généralement stockées dans des bases de données web qui sont souvent distribuées dans l'optique de mieux faire face à un nombre d'utilisateurs important (des centaines de millions). Avec la réplication, une même donnée peut être lue et/ou modifiée sur différents sites ce qui favorise le parallélisme et donc le passage à l'échelle. La réplication peut être effectuée suivant deux configurations possibles des sites qui stockent les copies des données : la configuration mono-maître et la configuration multi-maître.

La réplication mono-maître utilise un seul nœud maître sur lequel les requêtes de mise à jour sont effectuées alors que les requêtes de lecture sont envoyées sur les nœuds esclaves. Cette configuration permet de paralléliser uniquement les requêtes de lecture. La réplication multi-maître permet de paralléliser aussi bien les requêtes de mises à jour que les requêtes de lecture, car les mises à jour peuvent être exécutées sur plusieurs sites ainsi que les opérations de lecture. Cependant, répliquer des données dans un environnement p2p est un véritable défi puisque les pairs peuvent tomber en panne à tout moment. Ainsi, si plusieurs sites maîtres tombent en panne, le modèle de réplication multi-maître converge vers une réplication mono-maître, ce qui entraine la congestion de certains nœuds et le déséquilibrage des charges de mises à jour. De plus, la panne d'un nœud maître peut compromettre la cohérence des données ou entrainer la perte de mises à jour en cas de réplication asynchrone. Pour éviter l'occurrence des congestions, il faut contrôler le nombre de nœuds maître et s'assurer qu'il ne diminue pas en valeur relative. Pour éviter les pertes de mises à jour ou l'introduction de la cohérence, il faut surveiller la durée de vie des nœuds maîtres afin de les envoyer des mises à jour que lorsqu'ils sont suffisamment disponibles pour pouvoir les traiter.

Beaucoup de solutions utilisant la réplication multi-maître existent [9, 10, 11, 12, 14], mais aucune d'entre elles ne s'attaque au problème de gestion des nœuds maîtres dans l'optique de garder les performances du système en cas de déconnexion intempestives. Les travaux présentés dans ce papier ont pour but d'étudier ce problème crucial surtout pour des environnements à grande échelle.

Dans cette perspective, nous proposons des mécanismes de gestion des nœuds maîtres qui s'appuient sur la durée de vie des nœuds. La durée de vie d'un nœud peut être obtenue par estimation grâce à des modèles mathématiques [1] et est très utilisé pour caractériser la qualité d'un réseau P2P. De fait, connaissant la durée de vie restante d'un nœud et le temps nécessaire pour traiter une requête, nous vérifions si le nœud restera aussi longtemps dans le système pour finir une requête que l'on veut lui envoyer pour exécution. Cette vérification permet alors de pouvoir éviter l'interruption de l'exécution d'une requête et du coup de pouvoir borner le temps de réponse. Par ailleurs, la connaissance de la durée de vie d'un nœud sera également utilisée pour configurer dynamiquement les nœuds maîtres et esclaves. La déconnexion d'un nœud maitre pouvant compromettre la cohérence ou la disponibilité des mises à jour, il devient important de pouvoir changer un nœud maître par un nœud esclave dont la durée de vie est beaucoup plus importante. En résumé, l'approche proposée permet : (1) d'éviter d'envoyer des mises à jour vers un nœud qui va se déconnecter avant la fin de leurs exécutions ; (2) de définir la fréquence de propagation des mises à jour vers les nœuds esclaves; (3) de définir dynamiquement le statut d'un nœud en fonction de l'état du système (nombre de nœuds maitres restant et nombre de nœuds esclaves).

Les principales contributions de ce papier sont :

Un mécanisme permettant de remplacer un nœud maître par un nœud esclave. Il utilise un algorithme qui surveille et évalue la durée de vie restante des nœuds pour promouvoir un nœud esclave en maître afin de garder une bonne proportion des nœuds maître.

Un modèle de propagation des mises à jour entre nœuds maître et esclave suivant une fréquence déterminée à partir de la durée de vie estimée des nœuds considérés. Ce modèle présente l'avantage de réduire les risques de pertes de mises à jour dans le cas d'une réplication asynchrone et de minimiser la durée de la synchronisation puisque

seuls les nœuds en voie de déconnexion ou devant participer à la promotion d'un nœud esclave sont synchronisés.

Une évaluation des protocoles proposés avec l'utilisation du simulateur PeerSim et un banc d'essai pour quantifier les gains obtenus.

La suite de cet article est structurée comme suit. La section 1.1 présente l'architecture de notre système et la section 1.2 son fonctionnement. La section 2 détaille l'algorithme de remplacement et l'algorithme de choix du nœud esclave optimal. La section 3 présente les résultats de notre simulation et en fin à la section 4, nous concluons.

1.1 Architecture et problématique

Dans cette section, nous présentons l'architecture de notre système et décrivons les problèmes abordés.

1.1.1 Architecture et concepts de base:

Nous considérons un système P2P S_P, structuré en des super-pairs et des pairs simples. Les super-pairs jouent le rôle de serveur et les pairs simples le rôle de client. Cette topologie de réseau P2P, plus connue sous le nom de réseau semi-structuré ou hybride [15], permet de garder les avantages de l'architecture client serveur et ceux d'un réseau P2P purement décentralisé en facilitant la gestion et le contrôle des ressources d'une part et en favorisant la distribution des traitements d'autre part. L'ensemble des pairs simples connectés à un super-pair forme un cluster et sont souvent situés dans un même réseau LAN.

Par la suite, nous considérons une base de données partiellement répliquée sur S_P, avec une configuration multi-maître. De fait, les copies maîtres sont stockées sur les super-pairs de S_P et les copies esclaves sur les pairs simples de S_P. Chaque cluster contient un fragment de la base. Sur la Figure 1, les copies maîtres sont en rouge alors que les esclaves sont en noire. Chaque nœud maître est relié à un ensemble de nœuds esclaves qui se trouvent dans le même cluster que lui et qui stocke une copie du même fragment. Les nœuds maîtres sont également reliés via internet pour assurer la cohérence globale du système.

De plus, la charge applicative reçue est composée de deux types de requêtes : (1) les requêtes de lecture seule (*read-only*) qui accèdent aux données sans les modifier et (2) les requêtes de mises à jour (*update*) qui effectuent des modifications. Nous mentionnons que les requêtes de mises à jour sont toutes effectuées que sur les nœuds maîtres et les requêtes de lecture seules sont envoyées sur les nœuds les moins chargés (nœud esclave ou maître). Remarquons aussi que si un nœud maître reçoit une mise à jour, tous les autres nœuds, qu'ils soient dans le même cluster ou non, sont des esclaves vis à vis de ce nœud maître pour la modification en cours.

Pour garder la cohérence mutuelle, les transactions effectuées sur les nœuds maîtres sont propagées de manière séparée ultérieurement sur les autres nœuds esclaves. Ce modèle d'exécution est appelé réplication asynchrone et permet de réduire le temps de réponse. Notons que la propagation des mises à jour d'un nœud maître vers les nœuds esclaves se fait de manière progressive, c'est à dire un nœud maître n'envoie pas simultanément ses mises à jour à tous les autres nœuds esclaves. Par exemple, après la

validation d'une transaction de mise à jour, un nœud maître peut envoyer les mises à jour à un sous-ensemble ***A*** des nœuds esclaves à un instant λ pour des besoins spécifiques avant de les envoyer à un autre sous-ensemble ***B*** à l'instant λ'. En d'autres termes, à un instant λ les copies peuvent diverger car n'ayant pas toutes encore reçus les dernières mises à jour mais à terme toutes les copies auront le même état. Ce modèle de cohérence est appelée cohérence à terme et permet de passer plus facilement à l'échelle [16].

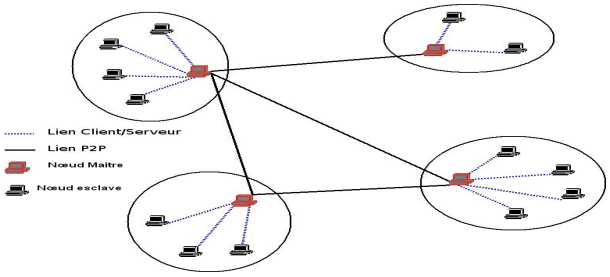

Figure 1 : *Architecture de notre système*

1.1.2 Positionnement du problème:

Soit une requête d'un client, qui connaît un certains nombre de nœuds maîtres. Pour traiter cette requête, le client doit lui envoyer à un des nœuds maîtres. Si c'est une requête de lecture, le nœud maître va regarder au niveau de son groupe le nœud esclave le moins chargé et lui envoie la requête. Si la requête est de type écriture elle sera exécutée par le nœud maître lui même.

Le problème auquel pourra face ce système est le suivant : un nœud maître peut se déconnecter à n'importe quel moment puisqu'on est dans un système P2P.

Cette déconnexion peut être volontaire ou involontaire. Ce qui peut entraîner les conséquences suivantes sur le réseau :

– des pertes de requêtes : les requêtes envoyées à un nœud maître en panne ne seront pas exécutées et donc le client ne recevra point de résultats ;

– une perte de cohérence : un nœud maître n'ayant pas encore propagé ses dernières mises à jour avant de tomber en panne entraine une incohérence au niveau des données.

– une diminution des nœuds maîtres : si plusieurs nœuds maîtres tombent simultanément en panne alors les nœuds maîtres restants supporteront toute la charge applicative, ce qui entraine des congestions.

L'objectif de ce papier est de pouvoir faire face à ces problèmes par anticipation. En d'autres mots, nous cherchons à trouver des stratégies pour garantir que les requêtes puissent être envoyées que sur des nœuds qui pourront les exécuter entièrement avant de se déconnecter. De plus, nous cherchons à trouver un mécanisme de gestion de nœuds maître afin d'en garder une quantité suffisamment importante pour éviter toute source de congestion.

2. Gestion dynamique des nœuds

Pour faire face aux problèmes cités précédemment, nous proposons un modèle dynamique de configuration des répliques de la base de données. Le principe de fonctionnement est simple : conférer le statut de nœud maître aux nœuds ayant une durée de vie restante beaucoup plus élevée. Ainsi, si la durée de vie d'un nœud maître tend vers zéro, il faut le remplacer par un nœud esclave avec une durée de vie très importante. Le problème ne se pose pas pour les nœuds esclaves car leurs déconnexions n'entrainent pas de pertes de mises à jour ou de cohérence. Bien que la panne des nœuds esclaves puisse avoir un impact sur les performances, il faut noter que leur nombre est tellement important que la panne de quelques uns d'entre est négligeable car n'entraîne pas des contentions. Pour connaître la durée de vie des nœuds, nous utilisons un modèle d'estimation que nous décrivons dans la section 2.1. Les algorithmes pour choisir un nœud à promouvoir en nœud maître et l'algorithme pour faire le remplacement sont décrits respectivement dans les sections 2.3 et 2.2.

2.1 Modèle d'estimation

Rappelons que notre approche, dans l'optique de pouvoir remplacer un nœud maitre sur le point de se déconnecter par un de ses nœuds esclaves, doit faire appel à l'estimation des durées de vie. Pour cela, nous considérons qu'un nœud maitre, doit toujours être informé des arrivées et départs de ces derniers. Ceci lui permet d'estimer leurs durées de connexion et de choisir judicieusement le nœud susceptible de le remplacer. Dans ce papier, nous utilisons l'une des techniques d'estimation de durée de vie proposées dans [1], nommément la méthode binomiale.

Cette méthode permet de prédire des quantiles des distributions de la disponibilité. Les quantiles sont des points essentiels pris à des intervalles réguliers d'une fonction de distribution d'une variable aléatoire qui n'est rien d'autre que la durée de vie d'un nœud dans notre étude.

Ce sont ces quantiles qui sont utilisés pour déterminer un minimum de disponibilité estimé en faisant la somme des probabilités d'obtenir une disponibilité supérieure au q^{me} quantile de chaque mesure de disponibilité.

2.2 Algorithme de remplacement

Comme nous l'avons déjà annoncé précédemment, un nœud maître dont la durée de vie s'épuise doit être remplacé par un nœud esclave. Pour ce faire, nous supposons qu'il existe toujours un nœud esclave disponible et nous supposons l'existence d'un seuil xs, à partir du quel on doit procéder au changement de statut d'un nœud maître en nœud esclave et vice versa. Si LT_m est la durée de vie restant d'un nœud maître, alors le changement aura lieu dès que $LT_m \leq xs$.

Pour procéder au changement de statut, le nœud maître vérifie la liste de ses nœuds esclaves disponibles et choisit le nœud esclave le plus optimal, c'est à dire le nœud dont le coût de remplacement est le plus faible. L'algorithme pour choisir le nœud le plus optimal est décrit dans la section 2.3.

Une fois, le nœud optimal identifié, le nœud maître doit effectuer les tâches suivantes :

– Arrêter d'exécuter les mises à jour après exécution de la transaction en cours.

– Envoyer un message de changement de statut aux nœuds esclaves de son cluster et aux nœuds maîtres distants. Ce message inclut toutes les informations relatives au nœud choisi, plus précisément les informations de connexion.

– Transférer la liste des nœuds esclaves du cluster au nœud promu.

– Mettre à jour ses informations de connexions avec les nœuds maîtres distants.

– S'il y a des mises à jour en attente d'exécution les router vers le nœud promu.

Notons que durant la période de remplacement, le nœud maître peut continuer à recevoir de nouvelles mises à jour qu'il va transférer au nouveau maître.

2.3 Algorithme de choix d'un nœud esclave

L'idée est de trouver le nœud esclave, N_i, le plus optimal pour le promouvoir en nœud maître en fonction de sa durée de vie ($\boldsymbol{LT_i}$) et de son obsolescence $\boldsymbol{O}$ (nombre de mises à jour manquantes). Comme on peut avoir un nœud qui a un $\boldsymbol{LT_i}$ plus grand que les autres mais dont le niveau d'obsolescence est très importante, alors le temps nécessaire pour rafraîchir ce nœud (appliquer les mises à jour manquantes) est supérieur à la durée de vie restante du nœud maître en voie de déconnexion. Pour éviter ce cas, le choix va se porter sur le nœud qui domine sur la durée de vie et sur la durée de rafraîchissement. Autrement, il s'agit du nœud dont la durée de vie restante sera la plus élevée une fois que l'opération de rafraîchissement est effectuée.

Pour ce faire, soit $\boldsymbol{T_{exec}}$, le temps moyen d'exécution d'une transaction T et $\boldsymbol{\delta_i}$ le temps nécessaire pour rafraîchir le $\boldsymbol{N_i}$, alors $\boldsymbol{\delta_i = O_i * T_{exec}}$. Le temps moyen d'exécution de T est obtenu en calculant la moyenne glissante des transactions qui ont précédé T.

Afin de choisir le meilleur nœud, le nœud maître doit calculer d'abord pour chacun des nœuds candidats son $\boldsymbol{LT_i}$ après exécution éventuelle des opérations de rafraîchissement (i.e $\boldsymbol{LT_i = LT_i - \delta_i}$). Puis, il choisit celui dont le $\boldsymbol{LT_i}$ est minimal. Formellement, si $\boldsymbol{NE}$ est la liste des nœuds candidats, $\boldsymbol{N_j}$ sera choisi si $\boldsymbol{\forall N_i \in NE, LT_i < LT_j}$.

3. Expérimentation et validation

L'objet de cette section est de vérifier la faisabilité de notre approche. Pour cela nous commencerons d'abord par juger l'intérêt d'avoir une bonne proportion de nœuds maître.

Puis nous comparons le gain obtenu par notre approche qui propose une gestion dynamique des nœuds maîtres par rapport à la solution classique qui s'appuie sur une configuration statique des nœuds maîtres. Ce gain sera évalué en terme de débit transactionnel et d'équilibrage des charges.

3.1 Paramétrage de notre simulation

Avec Peersim [2], nous avons effectué les expériences avec la configuration suivante : nous avons initialisé le système avec 45 nœuds maîtres et 400 nœuds esclaves liés à eux. Nous avons introduit dans le système des pannes involontaires ne dépassant pas 3% du nombre total de nœuds maîtres. Nous avons fait varier le nombre de mises à jour émises par les clients et nous avons mesuré la charge du système qui correspond au pourcentage d'exécution des transactions : nombre de transactions traitées /nombre de transactions émises. Pour simuler l'occurrence des déconnexions, nous diminuons progressivement le nombre de nœuds du système.

3.2 Impact du nombre de nœuds sur le traitement des transactions

La première série de tests effectués montre l'impact du nombre de nœud dans notre système. Nous évaluons la performance du système en vérifiant que la diminution du nombre de nœuds maître est source de congestion. Pour cela, nous fixons le nombre de nœud maître à 45 et nous le décrémentons jusqu'à 15, soit 66,66% d'occurrence de pannes des nœuds maître. Ce taux de pannes est suffisant pour pouvoir montrer l'impact du nombre de nœuds maîtres sur le traitement des transactions comme en atteste d'ailleurs la figure 2. Nous remarquons une faible diminution du débit transactionnel pour un pourcentage de panne faible, autrement, jusqu'à 33% de pannes de nœud maître, le débit transactionnel ne chute que de 20%. Au delà de ce pourcentage, nous avons une diminution drastique du débit d'environ 60%. Ce résultat révèle tout l'intérêt de garder une bonne proportion de nœuds maîtres en faisant de sorte que leur nombre ne décroît pas de manière significative (i.e. de plus de 33%).

3.3 Apport de notre proposition en termes de débit transactionnel

L'objectif de notre approche étant de garder une bonne proportion des nœuds maître en faisant une configuration dynamique des répliques du système. Pour évaluer le gain de la réplication avec configuration dynamique, nous la comparons avec un modèle de réplication qui se base sur une configuration statique des répliques.

Dans chacune des approches, nous avons attribué aux nœuds des durées de vie de 60 cycles, ce qui constitue la durée de nos expériences. Pour les besoins de cette expérience, nous fixons un seuil faible (5 cycles). Ce seuil définit le moment à partir duquel le remplacement doit avoir lieu. La charge applicative totale est composée de 50% de requêtes de lecture seule et de 50%de mises à jour. La figure 3 représente les résultats obtenus. Nous remarquons que la gestion dynamique présente un débit transactionnel beaucoup plus important que la gestion statique. Par exemple dans le cas extrême (près de 66% de pannes maître), notre approche outrepasse de plus de 35% l'approche statique en termes de débit transactionnel.

Ce gain s'explique par le fait que si un nœud maître se déconnecte, on le remplace par un autre nœud esclave, et donc permet de garder une bonne portion des nœuds maîtres. Cependant, cela a l'inconvénient de diminuer le nombre de nœuds esclave, raison pour laquelle, le débit transactionnel avec la gestion dynamique décroit légèrement quand le nombre de déconnexions croît. Avec l'approche statique, un nœud maitre n'est pas remplacé avant sa déconnexion et du coup, les mises à jour qu'il était

entrain de traiter seront perdues. De plus, la diminution des nœuds maître entraine des problèmes de congestion qui rallonge le temps de réponse.

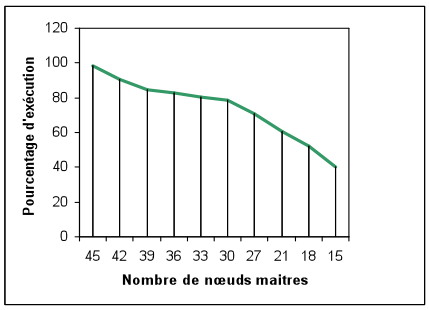

Figure 2. *Impact du nombre de nœuds maitres sur l'exécution des transactions*

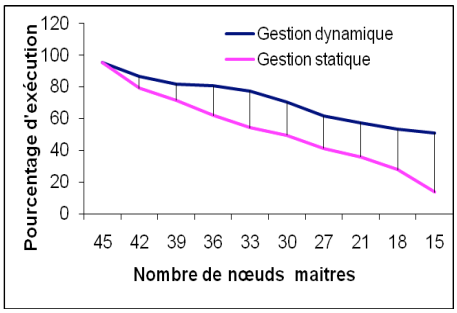

Figure 3. *Système de gestion dynamique et système de gestion statique des nœuds maitres*

3.4 Apport de notre proposition en termes d'équilibrage de charge

Outre le gain obtenu en termes de débit transactionnel, notre approche présente aussi l'avantage de favoriser l'équilibrage de charge. Pour montrer cela, nous comparons notre stratégie de routage des requêtes avec la méthode du tourniquet, plus connu sous le nom de round robin. Cette stratégie est utilisée pour répartir les requêtes sur des sites maîtres en supposant que ces derniers sont constants. C'est pourquoi nous l'utilisons avec la réplication statique. Notre stratégie, quant à elle s'appuie sur la durée de vie restante des nœuds. De ce fait, un nœud maître n'est choisi que s'il est disponible pour traiter la requête, autrement, il n'est pas chargé et il a une durée de vie restante importante (supérieure au temps de traitement des transactions). De même que l'expérience précédente, la charge applicative est composée de 50% de mises à jour et de 50% de requêtes de lecture. Nous mesurons le coefficient de variation $\boldsymbol{T}$ de l'équilibrage de charge. Ce coefficient est obtenu en divisant la déviation standard (δ) par la moyenne de charge $\boldsymbol{E}$, $\boldsymbol{T=\delta/E}$. Il permet de mesurer la déviation de la répartition actuelle des charges par rapport à une répartition parfaite (uniforme). En d'autres mots, plus la déviation est faible plus l'équilibrage des charges est parfait. Nous reportons dans la figure 4 les résultats de nos expériences et nous remarquons que le coefficient de variation avec notre approche est inferieur à celui associé à la stratégie round robin. Cela s'explique par le fait que l'envoie des requêtes est faite en prenant en compte aussi bien de la disponibilité des nœuds que de leur charges. Du coup une requête est toujours envoyée que vers la réplique la moins chargée et disposant suffisamment de temps de vie restante.

3.5 Surcharge de notre approche

L'objectif de cette expérience est de quantifier la surcharge générée par notre approche de gestion dynamique. De fait, durant le remplacement d'un nœud maître des messages sont envoyés aux autres nœuds et l'exécution des mises à jour sera ralentie car le nœud maître concerné est inutilisable pour cause de transfert des transactions de rafraîchissement et le nouveau maître n'est pas encore prêt pour exercer intégralement sa fonction. Intuitivement, plus la durée de vie des nœuds est faible, plus les échéances de remplacement seront importantes et plus la disponibilité des nœuds maître sera

réduites. Pour mesurer l'impact des multiples remplacements, nous faisons varier la durée de vie des nœuds de 55 à 5 cycles dans l'optique d'avoir plus de déconnexions.

Les résultats sont représentés dans la figure 5. Nous pouvons remarquer que quand la durée de vie est comprise entre 55 et 25 cycles le pourcentage d'exécution décroit très légèrement. Cependant à partir de 20 cycles nous remarquons une forte décroissance de la courbe. Cela s'explique par le fait que quand la durée de vie est faible, le système est face a beaucoup plus de remplacement et donc passe plus de temps à faire des opérations de synchronisation et de promotion que de traiter de nouvelles requêtes. Ce phénomène peut donc être évité en choisissant judicieusement de conférer le statut de nœud maître qu'aux pairs qui ont une durée de vie moyenne très élevée. Cette durée de vie moyenne ne peut être calculée qu'en observant le système pendant une période très longue et s'appuyant sur des modèles mathématiques puissants. Nos travaux en cours sont d'ailleurs inscrits dans cette dynamique afin de rendre notre solution très pratique.

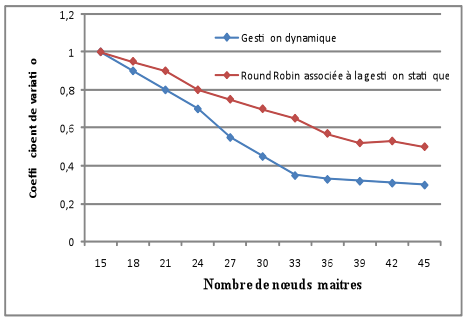

Figure 4. *Gestion dynamique et gestion statique associée à round robin*

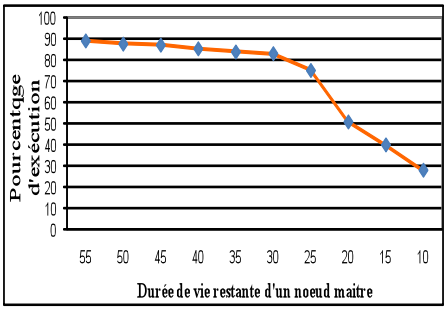

Figure 5. *Impact du nombre de remplacement sur notre approche*

4. Conclusion

Dans cet article nous avons proposé une solution permettant d'avoir une meilleure utilisation des ressources disponibles du système en s'appuyant sur la connaissance probable de la durée de vie des nœuds. Nous avons proposé un algorithme permettant de remplacer un nœud maître dont la durée de vie restante s'approche d'un certain seuil par un nœud esclave. De plus, nous proposons un algorithme de routage basé sur la durée de vie restante d'un nœud, ce qui permet de minimiser les pertes de mises à jour et garder un meilleur équilibrage des charges. Les résultats de nos expériences ont montré que le remplacement réduit non seulement les pertes de mises à jour mais aussi assure l'équilibrage de charge dans ces systèmes à haute volatilité. Elles ont aussi montré que le choix d'un bon seuil est utile afin de minimiser la surcharge induite par nos mécanismes de remplacement. Nos travaux en cours sont dirigés vers la recherche d'un seuil optimal et d'étudier en profondeur d'autre modèles d'estimations afin d'en choisir le meilleur qui nous permet de trouver les nœuds optimaux qui seront désignés comme maîtres.

5. Bibliographie

[1] John Brevik, Daniel Nurmi, Rich Wolski, “Automatic Methods for Predicting Machine Availability in Desktop Grid and Peertopeer Systems”

[2] http://sourceforge.net/projects/peersim

[3] G. et O. Gardarin. *Le Client-Serveur*. Editions Eyrolles, 1997

[4] J.-C. Laprie. Dependable computing and fault tolerance: concepts and terminology. In Proceedings of the 15th International Symposium on Fault-Tolerant Computing (FTCS '85), pages 2.11, Ann Arbor, MI, June 1985.

[5] M. Bertier, O. Marin, and P. Sens. Implementation and performance evaluation of an adaptable failure detector. In Proceedings of the International Conference on Dependable Systems and Networks (DSN '02), pages 354.363, Washington, DC, June 2002.

[6] Ion Stoica, Robert Morris, David Karger, Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable Peer-To-Peer lookup service for internet applications. In Proceedingsof the 2001 ACM SIGCOMM Conference.

[7]Frank Dabek, Frans Kaashoek, David Karger, Robert Morris, andIon Stoica, “WIde -area Cooperative Storage with CFS,” in Proc. ACM SOSP, Banff, Canada, 2001.

[8] Ananth Rao, Karthik Lakshminarayanan, Sonesh Surana, Richard Karp, and Ion Stoica, “Load Balancing in Structured P2P Systems,” in Proc. IPTPS, Feb. 2003.

[9] C. Coulon, E. Pacitti, and P. Valduriez. Consistency management for partial repli c ation in a high performance database cluster. In ICPADS ’05 : Pro- ceedings of t he 11th International Conference on Parallel and Distributed Systems, pages 809–815, 2005.

[10] S. Gançarski, H. Naacke, E. Pacitti, and P. Valduriez. The leganet sys- tem : Freshness-aware transaction routing in a database cluster. Journal of Information Systems, 32(2) :320–343, 2007.

[11] I. Sarr, H. Naacke, and S. Gançarski. Transpeer : Adaptive distributed transaction monitoring for web2.0 applications. In SAC, pages 423–430, 2010.

[12] Idrissa Sarr, Hubert Naacke, and Stéphane Gançarski. Failure-tolerant tran- saction routing at large scale. In DBKDA, pages 165–172, 2010.

[13] M. Gueye, I. Sarr, and S. Ndiaye. Database replication in large scale sys- tems : optimizing the number of replicas. In EDBT/ICDT ’09 : Proceedings of the 2009 EDBT/ICDT Workshops, pages 3–9. ACM, 2009.

[14] M. Patino-Martinez, R. Jimenez-Peres, B. Kemme, and G. Alonso. MIDDLE-R, Consistent Database Replication at the Middleware Level. ACM Transactions on Computer Systems, 28(4) :375–423, 2005.

[15] LIANG (J.), KUMAR (R.) et ROSS (K.W.). « Understanding KaZaA », 2004.

[16] W. Vogels. Eventually consistent. Commun. ACM, 52(1) :40–44, 2009