THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par *l'Université Toulouse III - Paul Sabatier*
Discipline ou spécialité : *Informatique*

Présentée et soutenue par Mahmoud EL SAMAD
Le *14 décembre 2009*

Titre : *Découverte et monitoring de ressources pour le traitement de requêtes dans une grille de données*

JURY

Djamal BENSLIMANE	Professeur à l'Université Lyon I (LIRIS)	*Rapporteur*
Abdelmalek BENZEKRI	Professeur à l'Université Toulouse III (IRIT)	*Examinateur*
Abdelkader HAMEURLAIN	Professeur à l'Université Toulouse III (IRIT)	*Directeur de Thèse*
Nouredine MELAB	Professeur à l'Université Lille I (LIFL)	*Rapporteur*
Franck MORVAN	Maître de Conférences, Habilité à Diriger des Recherches à l'Université Toulouse III (IRIT)	*Invité*
Farouk TOUMANI	Professeur à l'Université Clermont-Ferrand II (LIMOS)	*Invité*

École doctorale : *École Doctorale Mathématiques Informatique Télécommunications de Toulouse*
Unité de recherche : *Institut de Recherche en Informatique de Toulouse*
Équipe d'accueil : *Pyramide*
Directeur de Thèse : *Abdelkader HAMEURLAIN*

Résumé : La gestion des données réparties en environnement de grille de données pose de nouveaux problèmes et présente un réel défi : la découverte de ressources, l'allocation de ressources, la réplication, les services de monitoring pour l'optimisation de requêtes …etc. Les systèmes de grille diffèrent principalement des systèmes parallèles et distribués par la grande échelle et l'instabilité (i.e. la dynamicité des nœuds). Dans cette thèse, nous nous intéressons à la phase de découverte de ressources pour l'évaluation efficace de requêtes réparties en environnement de grille de données. Nous effectuons d'abord un état de l'art sur les principaux travaux de recherche portant sur la découverte de ressources en se focalisant sur les critères importants (e.g. passage à l'échelle, découverte fiable, faible coût de maintenance) pour la découverte de sources de données qui est spécifique à un environnement de grille de données. Dans cette perspective, nous proposons ensuite une méthode de découverte de sources de données, basée sur l'utilisation des Tables de Hachage Distribuées (THDs), permettant un accès permanent en présence de la dynamicité des nœuds de n'importe quel nœud d'une Organisation Virtuelle OV_{locale} à toute autre OV_i ($i \neq locale$) dans le système, avec un faible coût de maintenance entre les THDs. Après la découverte de ressources, il est très important d'observer l'état actuel de ressources surtout que ces dernières sont partagées à une grande échelle, afin de prendre des décisions sur le choix du nœud d'exécution d'une jointure (ou d'une partie d'une jointure) par exemple. L'observation ou le monitoring de ressources peut être effectué pendant l'allocation initiale ou l'exécution. Dans ce contexte, nous proposons une méthode permettant la prise en compte de la variation des valeurs des paramètres hôtes et réseaux, pendant l'exécution, dans le calcul du temps de réponse d'une opération relationnelle. L'idée est alors d'intégrer les informations de monitoring dans un modèle d'exécution à base d'agents mobiles développé au sein de notre équipe. Enfin, nous validons nos propositions par une évaluation des performances.

Mots-Clés : Grille de données, Évaluation de requêtes, Découverte de ressources, Monitoring de ressources, Évaluation des performances.

Aux âmes de ma grande-mère Nawal et mon père Bassam,

A ma mère Fatmé, mon beau-père Naïm, ma sœur Diala et mon beau-frère Jalal,
pour leur soutien, leur tendresse et leur amour infini

Et à tous ceux que j'aime.

REMERCIEMENTS

Tout d’abord, je tiens à remercier Monsieur Luis Fariñas del Cerro, Directeur de l'Institut de Recherche en Informatique de Toulouse IRIT, pour m'avoir accueilli au sein de son laboratoire.

Mes remerciements s’adressent à Messieurs les Professeurs Djamal Benslimane et Nouredine Melab pour avoir lu d’une manière profonde mon document de thèse. Les diverses remarques parvenues ont contribué, sans doute, à l’amélioration du manuscrit. Aussi, j'adresse mes remerciements à Messieurs les Professeurs Abdelmalek Benzekri et Farouk Toumani, d’avoir accepté de participer au jury et pour l’examen de ma thèse.

Je suis très reconnaissant envers Monsieur le Professeur Abdelkader Hameurlain, mon directeur de recherche qui m’a appris, avec beaucoup de patience et de rigueur le métier de chercheur. Ses précieux conseils scientifiques et son soutien moral dans les moments difficiles m'ont permis de bien réussir ma thèse. Merci également, à Monsieur Franck Morvan, Maître de Conférences, Habilité à Diriger des Recherches qui m’a encadré et qui a suivi les moindres détails de mes travaux de recherche. Merci d’avoir trouvé les bons mots pour m’encourager et de m’avoir permis de donner le meilleur de moi-même. Merci à Monsieur Riad Mokadem, Maître de Conférences, pour toutes les discussions intéressantes partagées et pour sa disponibilité.

Je remercie tous les membres de l'équipe Pyramide de la bonne ambiance, du plaisir d’avoir partager un café tous les matins, et surtout de m’avoir écouter quand j’en avais besoin.

Toute ma reconnaissance va envers ma famille pour leur aide durant mes longues et chères années d’étude en France.

Enfin, un grand merci pour ma seconde famille à Toulouse. Bien sûr, je parle de mes amis qui ont rendu ma vie plus agréable dans la ville rose.

Un dernier merci pour toutes les personnes que j’ai rencontrées durant mon doctorat, et à qui j’ai essayé d’expliquer le contenu de ma thèse et qui m’ont écouté avec gentillesse.

TABLE DES MATIÈRES

LISTE DES FIGURES

LISTE DES TABLEAUX

Chapitre I. Introduction

1. Contexte

De nos jours, la technologie de la grille Informatique a permis le développement des services permettant un partage efficace des ressources de calcul (e.g CPU, E/S) entre plusieurs organisations virtuelles (OVs) [Fos02]. Une OV est constituée d'un grand nombre d'utilisateurs (e.g. des scientifiques, des chercheurs), d'un très grand nombre de ressources (e.g. bases de données, CPU, mémoire, programme, services) et elle est souvent dédiée à un domaine d'application (e.g. la pathologie, la biologie, la physique) [Fos01, Iam01, Mas05, Tal05]. Dans un tel contexte, la gestion efficace de ressources est fondamentale pour les applications utilisant un système de grille.

La gestion de ressources joue un rôle très important dans un système de grille [Sch04] car elle inclut les phases suivantes : (i) la découverte de ressources, (ii) la sélection de ressources et (iii) l'allocation de ressources. Elle inclut aussi une étape importante qui est (iv) le monitoring de ressources qui peut intervenir pendant l'allocation et l'exécution. La découverte de ressources (i) consiste à rechercher les métadonnées décrivant les ressources pour l'exécution d'une tâche (e.g. une opération relationnelle)[1]. La sélection de ressources (ii) consiste à sélectionner un sous-ensemble de ressources parmi l'ensemble de ressources découvertes (e.g. si la phase de découverte de ressources retourne cent nœuds, alors la sélection de ressources consiste à réduire cet ensemble à une dizaine de nœuds par exemple, en fonction de la tâche soumise). L'allocation de ressources (iii), ou le placement de tâches, consiste à allouer le meilleur ensemble de ressources pour l'exécution d'une tâche (e.g. une opération relationnelle). Enfin, l'étape de monitoring de ressources (iv) consiste à observer l'état actuel de ressources afin de prendre une

[1] Dans la suite du document, nous employons la découverte de ressources pour désigner la découverte de métadonnées décrivant les ressources. Nous revenons en détail sur la terminologie employée dans le chapitre suivant.

décision en fonction de cette observation (e.g. on peut changer le nœud initialement alloué pour l'exécution d'une tâche suite à une surcharge CPU de ce nœud vers un autre nœud dont la CPU est moins chargée).

Dans cette thèse, nous nous intéressons à la phase de découverte de ressources considérée comme « la clé du succès » d'un système de grille [Ham08] et à l'étape de monitoring de ressources, pour l'évaluation de requêtes en environnement de grille de données. La grille de données diffère de la grille de calcul par la manipulation des données de plus en plus volumineuses, hétérogènes et réparties dans plusieurs OVs. En effet, les grilles de calcul sont utilisées pour des applications nécessitant des calculs intensifs et où la gestion de données est effectuée en se basant sur des solutions basiques d'échange de fichiers comme le GridFTP [Gri02] ou le GridNFS [Hon05]. Bien évidemment, ces solutions sont mal adaptées en environnement de grille de données.

Quoi qu'il en soit, les Systèmes de Gestion de Base de Données (SGBD) offrent des grandes facilités pour la gestion des données en environnement de grille de données et un support pour le stockage et l'analyse des données d'applications [Din01]. Dans ce contexte, les techniques utilisées dans les SGBD distribués et parallèles sont étendues puis combinées avec celles utilisées dans les systèmes de grille [Smi02, Alo03, Alp03, Ant05]. L'objectif de cette combinaison est de permettre un traitement de requêtes distribuées avec un langage de requêtes de haut niveau comme le SQL. Le traitement de requêtes SQL dans un système de grille est beaucoup plus compliqué par rapport au traitement de requêtes dans les systèmes de bases de données distribuées [Gar90, Ozs99] et parallèles [Ham96] vu les propriétés de la grande échelle et de l'instabilité d'un système de grille [Ham08]. La grande échelle signifie un très grand nombre d'utilisateurs et de ressources (e.g. ressources de calcul, sources de données, programmes, services) qui sont hétérogènes, autonomes et réparties sur plusieurs nœuds. L'instabilité du système (i.e. la dynamicité des nœuds) signifie qu'un nœud[2] peut joindre ou quitter (e.g. suite à une panne) le système à tout instant.

La phase de découverte de ressources est une phase très importante pour l'évaluation de requêtes en environnement de grille de données car elle consiste à découvrir les métadonnées décrivant les ressources (e.g. données, CPU, bande passante) nécessaires pour l'exécution d'une

[2] Un nœud peut être une machine monoprocesseur, une station de travail, une machine parallèle, un serveur ou une grappe de machines …etc. accessible par un seul point d'accès,

requête. Dans un contexte de gestion de données en environnement de grille de données, on peut distinguer deux types de ressources : les sources de données (e.g. des relations) et les ressources de calcul (e.g. la CPU d'un nœud, la bande passante entre deux nœuds). Deux phases interviennent alors durant l'évaluation d'une requête *Req* : (i) la phase de découverte de sources de données[3] référencées dans *Req* et (ii) la phase de découverte de ressources de calcul[4] possiblement allouables pour l'exécution de l'ensemble des opérations relationnelles (e.g. une jointure ou une partie d'une jointure) de *Req*. Dans la phase (i), il est indispensable de découvrir les métadonnées décrivant toutes les sources de données référencées dans *Req* (au pire, il faut au moins découvrir un réplica d'une source de données référencée dans *Req* pour que l'exécution soit possible sinon il est impossible d'exécuter *Req* et un message d'échec est envoyé à l'utilisateur). La phase (ii) consiste à découvrir les métadonnées décrivant un ensemble de ressources de calcul suivant un ensemble de critères (e.g. un nœud dont la vitesse CPU >1 Ghz et valeur de la charge CPU < 50% et une taille de la RAM > 512 Mo) pour l'exécution des opérations relationnelles de *Req*. La phase (i) est spécifique aux grilles de données quant à la phase (ii), elle a été largement étudiée dans les grilles de calcul [Iam01, Che05, And02, Cai03, Bha05, Gao04, Jea08, Spe03, Ram04, Sch03, Opp04, …etc.]. Les métadonnées découvertes dans les deux phases (i) et (ii) sont indispensables pour les prochaines phases de l'évaluation de requêtes (e.g. l'optimisation).

Dès lors que les métadonnées de ressources sont découvertes, une phase d'agrément[5] doit être effectuée pour vérifier si ces ressources sont bien accessibles. Ensuite, l'optimiseur doit effectuer la sélection des meilleures réplications de sources de données (e.g. relations) parmi l'ensemble de sources de données découvert, la sélection des meilleures ressources de calcul parmi l'ensemble des ressources découvertes, le placement des opérateurs relationnels (appelé aussi allocation de ressources) …etc. Afin de mieux prendre ces décisions, il est primordial d'observer l'état actuel de ressources dans un système de grille car ces ressources sont partagées à échelle mondiale (e.g. la taille d'une relation, la valeur de la charge CPU d'un nœud, la valeur de la charge de la bande passante entre deux nœuds), pour permettre une exécution efficace des requêtes SQL. L'observation ou le monitoring peut intervenir pendant l'allocation initiale [Gou06, Soe05,

[3] La découverte de sources de données désigne la découverte de métadonnées décrivant les sources de données
[4] La découverte de ressources de calcul désigne la découverte de métadonnées décrivant les ressources de calcul
[5] L'agrément signifie qu'on vérifie si une ressource est bien connectée au système et qu'il est possible d'atteindre cette ressource

Liu08]. Il s'agit d'un scénario classique d'utilisation des informations de monitoring. Le monitoring peut aussi intervenir pendant l'exécution [Sam08, Woh07]. Les valeurs d'un certain nombre de paramètres (e.g. valeur de la charge CPU d'un nœud, valeur de la charge de la bande passante entre deux nœuds, taille des données) sont observées pendant l'exécution. Ensuite, l'optimiseur peut décider, par exemple, de changer le nœud initialement alloué (pour une opération relationnelle) suite à une surcharge de la CPU de ce nœud, vers un autre nœud dont la valeur de la charge CPU est moins importante. Dans un tel contexte, les méthodes d'optimisation dynamiques permettent d'adapter les plans d'exécution aux changements de paramètres de l'environnement d'exécution. En particulier, nous nous intéressons à l'étude de la variation des valeurs des paramètres hôtes (i.e. les valeurs de la charge CPU, de la charge des E/S et de la charge mémoire d'un nœud) et réseaux (i.e. les valeurs de la latence et de la charge de la bande passante entre deux nœuds) pour l'optimisation dynamique de requêtes en environnement de grille de données. Ainsi, pour pouvoir capturer la variation des valeurs des paramètres hôtes et réseaux, il est opportun d'utiliser un outil de monitoring de ressources de calcul capable d'observer l'état actuel de ces ressources et de calculer la meilleure valeur estimée d'un paramètre.

Dans la suite, nous traitons la problématique, l'objectif et les contributions pour la partie découverte de ressources puis pour la partie monitoring de ressources.

2. Découverte de ressources

2.1. Problématique et objectif

Dans les systèmes de bases de données distribuées, le traitement d'une requête SQL procède en quatre phases [Gar90] : la décomposition, la localisation de données, l'optimisation et l'exécution. Les deux premières phases utilisent les métadonnées stockées dans le schéma global. Ce dernier est constitué de deux parties : (i) le schéma conceptuel global et (ii) le schéma de placement. Le schéma conceptuel global (i) définit toutes les données appartenant à la base de données réparties. Le schéma de placement (ii) précise comment les données sont placées sur les différents nœuds du système. Il contient toutes les métadonnées relatives à la localisation, la fragmentation et la duplication des données.

Dans la littérature [Gar90, Ozs99], le schéma global peut être organisé selon trois approches principales : centralisée, dupliquée et répartie. Dans l'approche centralisée, le schéma global est

stocké dans un nœud central. La centralisation peut créer un goulet d'étranglement si un grand nombre de messages est envoyé vers ce nœud central. D'autre part, la panne du nœud central rend la base de données inaccessible et inutilisable. Dans l'approche dupliquée, chaque nœud contient une copie du schéma global évitant ainsi les inconvénients de l'approche centralisée. La mise à jour du schéma global doit être effectuée sur tous les nœuds pour conserver la cohérence ce qui peut rapidement inonder le réseau dans un système à grande échelle. Les deux approches précédentes (i.e. centralisée et dupliquée) ne passent pas à l'échelle [Pac07]. Le schéma global doit être organisé d'une manière complètement répartie en environnement de grille de données. Un facteur important apparaît qui se résumé en, comment répartir et accéder aux métadonnées désirées d'une manière efficace. Dans ce contexte, nous nous sommes intéressés à la conception d'une méthode distribuée efficace de découverte de sources de données référencées dans une requête SQL. La découverte de sources de données est un vrai défi dans des OVs instables et à grande échelle. Dans ces OVs, les utilisateurs (e.g. des chercheurs, des scientifiques) partagent leurs sources de données pendant une durée déterminée et peuvent joindre ou quitter leur OV à tout moment.

Dans ce contexte, les méthodes de découverte de ressources basées sur des techniques pair-à-pair (P2P) sont proposées. Ces méthodes permettent un meilleur passage à l'échelle [Pac07, Tal07a, Tru07, Mes08, Ran08] que la plupart des méthodes actuelles de découverte de ressources (i.e. les méthodes basées sur une approche client/serveur [Ant05, Che00, Dee04, Fit97, Hua04, Al-H05, Jay04, Kau07, Lyn09, Ram06, Sch06], les méthodes basées sur des agents [Cao01, Cao02, Din05, He05, Kak06, Zhu06, Yan07]). Les méthodes de découverte de ressources, basées sur des techniques P2P, peuvent être classées en trois catégories [Ham08, Pac07] : (i) les méthodes basées sur des techniques P2P non structurées [Abd05, Iam01, Iam04, Jea08, Tal05], (ii) les méthodes basées sur des techniques super-pairs [Fil04, Mas05, Pup05] et (iii) les méthodes basées sur des techniques P2P structurées [Ali07, Dov03, Gal03, Xia05].

Dans les méthodes basées sur des techniques P2P non structurées (i), la découverte de ressources est effectuée par diffusion ce qui peut créer rapidement une inondation dans un système constitué d'un très grand nombre de nœuds. L'utilisation d'un TTL (Time To Live)

[Iam01, Iam04] permet de réduire considérablement l'espace de la découverte mais ne garantit pas une découverte fiable[6].

Dans les méthodes basées sur des techniques super-pair (ii), la panne d'un super-pair peut rendre une partie du système (e.g. une OV) inaccessible et inutilisable.

Les méthodes basées sur des techniques P2P structurées (iii) utilisent les Tables de Hachage Distribuées (THDs). Ces méthodes sont les plus adaptées pour la découverte de sources de données car les THDs permettent de : (i) passer à l'échelle, (ii) fonctionner en présence de la dynamicité des nœuds et (iii) garantir une découverte fiable. Les THDs nécessitent un coût de maintenance assez élevé en présence de la dynamicité des nœuds. En particulier, ce coût peut être très élevé en présence d'un effet « churn »[7].

L'objectif de cette partie est : (i) de proposer une architecture d'évaluateur de requêtes SQL en environnement de grille de données (afin de mieux situer la phase de découverte de sources de données et de ressources de calcul, et l'étape de monitoring de ressources) (ii) de concevoir et développer une méthode efficace de découverte de sources de données ayant un faible coût de maintenance entre les THDs en présence de l'effet churn, et (iv) montrer la validité et la viabilité de la méthode proposée dans (ii) par des simulations.

2.2. Contributions

Nous proposons une méthode de découverte de sources de données [Sam09] pour l'évaluation de requêtes en environnement de grille de données. L'idée est d'associer une THD_i pour chaque OV_i du système où chaque OV_i est dédiée à un domaine d'application. Cette THD_i est responsable de la gestion des métadonnées de toutes les sources de données de l'OV_i. Deux niveaux de découvertes sont ainsi défini : intra-OV et inter-OV. Lorsqu'un utilisateur soumet une requête de découverte, la découverte intra-OV est d'abord lancée car nous favorisons le principe de la localité de données (qui signifie que les utilisateurs d'une OV accèdent souvent à des sources de données dans leur domaine d'application). Dans ce cas, il s'agit d'une découverte classique basée sur les THDs [Gal03, Kin07, Sto01, Row01]. Si la découverte locale échoue alors la découverte inter-OV est effectuée grâce à un système d'adressage permettant d'associer pour chaque nœud N d'une OV_{locale} un point de sortie (i.e. un nœud) N_i vers tout autre OV_i ($i \neq locale$) du système. Les

[6] La découverte fiable signifie qu'on est toujours capable de renvoyer une réponse à une requête de découverte en précisant si une ressource est bien présente dans le système (e.g. préciser si la relation « Alzheimer » figure bien dans le système)

[7] Le processus continu d'entrée et de sortie des nœuds [Rhe04, Wu06]

points de sortie vers une OV_i ne sont pas tous identiques afin d'éviter les inconvénients des nœuds centraux (i.e. le goulet d'étranglement si un grand nombre de messages est soumis à un nœud central et la panne de ce nœud peut causer le dysfonctionnement d'une partie du système). La méthode proposée garantit un accès permanent, en présence de la dynamicité des nœuds, de n'importe quel nœud d'une OV_{locale} à toute autre OV_i ($i \neq locale$). Le système d'adressage possède un faible coût de maintenance en présence de l'effet churn. Ce système est mis à jour lors de la découverte inter-OV.

3. Monitoring de ressources

3.1. Problématique et objectif

Dans cette partie, nous étudions l'impact de l'utilisation d'un service de monitoring pour un modèle d'exécution à base d'agents mobiles [Arc04, Hus05a, Ham06, Mor03]. Dans ce modèle, chaque opérateur relationnel (nommé opérateur mobile) est exécuté au moyen d'un agent mobile. Ce dernier réagit, en cours d'exécution, aux erreurs d'estimation et aux variations de valeurs de paramètres hôtes et réseaux en migrant d'un nœud à un autre. La décision de migration d'un agent est choisi en fonction de plusieurs métriques (e.g. le coût de production local des opérandes, le coût de migration d'un agent). Ces différentes métriques sont calculées en fonction du profil des opérandes et du résultat, et en fonction de la valeur des paramètres hôtes et réseaux dans le système. Dans la littérature, plusieurs approches ont été proposées pour la définition des modèles de coûts permettant l'estimation du coût des opérations : l'approche historique [Ada96], l'approche par calibration [Du95, Gar96] et l'approche générique [Naa99]. Dans ces approches, le modèle de coûts utilise des valeurs pour des paramètres hôtes et réseaux qui peuvent être obsolètes, imprécises ou incomplètes. En effet, le problème principal dans ces approches est que la valeur d'un paramètre est calculée à un instant *t1* et cette valeur est supposée être identique à l'instant *t2* ($t1 < t2$).

Étant donné le grand nombre de ressources partagées en environnement de grille de données, les valeurs des paramètres hôtes et réseaux sont indispensables afin de mieux estimer le coût de traitement d'une opération relationnelle (e.g. une jointure ou une partie d'une jointure). Le calcul de la valeur de ces paramètres nécessite un outil de monitoring capable d'estimer la meilleure valeur d'un paramètre hôte et réseau. Ainsi, il devient primordial de permettre au modèle de coûts d'accéder aux valeurs de paramètres hôtes et réseaux, à jour, dans un système de grille.

L'objectif de cette partie est : (i) proposer une méthode permettant la prise en compte de la variation de valeurs de paramètres hôtes et réseaux, pendant l'exécution, dans le calcul du temps d'exécution d'une opération relationnelle, (ii) étudier l'impact de l'utilisation d'un outil de monitoring pour un modèle d'exécution à base d'agents mobiles [Arc04, Ham06, Hus05a, Mor03] et (iii) de montrer la validité de la méthode proposée dans (i) par des expériences sur un réseau relié de nœuds d'exécution par un réseau local puis à grande échelle dans le cadre du projet GGM[8].

3.2. Contributions

Nous proposons une méthode [Sam08] permettant la prise en compte de la variation des valeurs des paramètres hôtes et réseaux dans le calcul du temps de réponse d'une opération relationnelle. La méthode proposée est basée sur l'utilisation des informations du monitoring pendant l'exécution. En particulier, nous nous intéressons à l'opérateur relationnel de la jointure mobile par hachage [Arc04]. Cette méthode permet aux agents mobiles une meilleure prise de décision lors de leur migration grâce à l'utilisation des valeurs de paramètres hôtes et réseaux à jour calculées avec un outil de monitoring. Nous étudions aussi l'impact des paramètres hôtes et réseaux sur le comportement de l'opérateur de la jointure mobile par hachage.

4. Organisation

Ce manuscrit est constitué de six chapitres. Dans le chapitre II, nous présentons un état de l'art sur les principales méthodes de découverte de ressources dans un système de grille et nous définissons un ensemble de critères cruciaux pour la conception d'une méthode efficace de découverte de ressources dans un système de grille. Nous précisons aussi l'ensemble de critères spécifiques à la découverte de sources de données pour l'évaluation de requêtes en environnement de grille de données.

Le chapitre III a pour objectif de proposer une architecture d'évaluateur de requêtes SQL en environnement de grille de données. Le but de cette architecture est de situer la phase de la découverte de sources de données, la phase de la découverte de ressources de calcul et l'étape de monitoring de ressources de calcul. Nous montrons aussi l'interaction entre ces différentes phases avec les différents modules et services présents dans cette architecture. Ensuite, nous proposons

[8] Projet 'ACI masse de données 2004', numéro 04 2 32

une méthode pour la découverte de sources de données environnement de grille de données. L'idée est d'associer une THD_i pour chaque OV_i du système où chaque OV_i est dédiée à un domaine d'application. La méthode proposée permet un accès permanent d'un nœud d'une OV_{locale} à toute autre OV_i ($i \neq locale$) avec un faible coût de maintenance entre les THDs en présence de l'effet churn.

Dans le chapitre IV, nous proposons une méthode permettant la prise en compte de la variation des valeurs des paramètres hôtes et réseaux dans le calcul du temps de réponse d'une opération relationnelle. Nous proposons ensuite une solution basée sur l'utilisation des informations de monitoring pendant l'exécution et nous justifions le choix d'un outil adapté à notre contexte, le Network Distant Service (NDS) [Gos07a]. Une formule de coûts permettant d'estimer le coût d'exécution d'une jointure mobile en prenant en compte la variation des valeurs des paramètres hôtes et réseaux est aussi présentée. Enfin, nous montrons comment l'intégration des informations de monitoring est effectuée pour la jointure mobile en détaillant les interactions entre un agent, le nœud local où il se trouve et NDS.

Le chapitre V est consacré à l'évaluation des performances de nos propositions. Nous évaluons, d'abord, la méthode de découverte de sources de données par simulation car nous ne disposons pas d'un accès à un grand nombre de nœuds. Ensuite, nous évaluons par exécution l'apport d'un service de monitoring pour la jointure mobile par hachage en fonction des valeurs des paramètres hôtes et réseaux.

Enfin, le chapitre VI établit une synthèse du travail proposé dans ce manuscrit et décrit les travaux futurs.

Chapitre II. Découverte de ressources dans un système de grille

1. Introduction

La communauté de la grille informatique s'est de plus en plus intéressée à la phase de découverte de ressources considérées comme la « clé du succès » d'un système de grille. En effet, la conception de méthodes efficaces de découverte de ressources est devenue un vrai challenge vu les deux caractéristiques principales d'un système de grille [Ham08]: la grande échelle et l'instabilité (i.e. la dynamicité des nœuds).

Une ressource peut consister en une ressource de calcul (e.g. CPU, E/S, bande passante), une source de données (e.g. une base de données, fichiers XML) ou un programme …etc. Quelque soit le type de la ressource, cette ressource (e.g. CPU) est décrite par un ensemble de métadonnées (e.g. vitesse CPU, taille du cache CPU, charge CPU). Nous distinguons entre deux types de métadonnées : les métadonnées statiques et les métadonnées dynamiques. Les métadonnées statiques sont les métadonnées qui peuvent changer de valeur occasionnellement (e.g. le système d'exploitation d'un nœud, le nom d'une relation, la vitesse CPU d'un nœud). Quant aux métadonnées dynamiques, elles concernent les métadonnées qui peuvent changer de valeur assez souvent (e.g. valeur de la charge CPU d'un nœud, valeur de la bande passante entre deux nœuds).

Ces métadonnées doivent être d'abord publiées pour permettre leur découverte plus tard [Dee04]. Montrons par un exemple très simple de notre vie quotidienne comment la publication et la découverte de métadonnées sont effectuées dans les systèmes d'information classique. : Un client *C* veut s'abonner auprès de France Telecom (FT) pour une ligne téléphonique. Ici, la ressource considérée est la ligne téléphonique décrite par un ensemble de métadonnées (e.g. numéro de la ligne, nom du bénéficiaire, adresse du bénéficiaire). D'abord, FT met à jour son

système d'information (e.g. un annuaire) en rajoutant les métadonnées ci-dessus. Cette mise à jour peut être appelée « publication de métadonnées ». Maintenant, une personne *P* peut consulter l'annuaire de FT pour des informations concernant *C* (e.g. son numéro de téléphone). Cette consultation peut être appelée « découverte de métadonnées ». Ensuite, la personne *P* peut directement contacter le client *C* grâce au numéro de téléphone trouvé dans l'annuaire. Dans un système de grille, la publication et la découverte de métadonnées sont plus compliquées que la mise à jour et la consultation d'un simple annuaire voire d'un moteur de recherche (e.g. Google). Une requête de découverte peut consister en plusieurs recherches simultanées (ou séquentielles) dans plusieurs OVs instables constituées d'un très grand nombre de ressources.

La découverte de ressources peut être définie [Iam01] comme suit : « étant donné une description de ressources, un mécanisme de découverte de ressources doit retourner un ensemble d'adresses de ressources qui correspond à cette description ». Les auteurs de [Tru07] définissent la découverte de ressources comme « le service permettant la localisation de ressources parmi plusieurs domaines administratifs suivant un ensemble d'attributs». Nous définissons le processus de découverte de ressources dans un système de grille comme suit : « c'est le processus responsable de trouver les métadonnées décrivant les ressources, issues d'une requête de découverte, dans plusieurs OVs ». En général, la plupart des travaux sur la découverte de ressources traitent plus précisément la découverte de métadonnées décrivant les ressources. Ainsi, dans la suite du document, la découverte de ressources est utilisée pour désigner la découverte de métadonnées décrivant les ressources.

Les méthodes de découverte de ressources peuvent être classées en trois catégories [Ham09, Sad08, Mes08, Ran08, Sed08, Tru07]: les méthodes basées sur une approche client/serveur [Ant05, Che00, Cza01, Dee04, Fit97, Hua04, Jay04, Kau07, Pas08, Ram06, Sch06], les méthodes basées sur des agents [Cao01, Cao02, Din05, Fuk06, He05, Kak06, Yan07, Yu06, Zhe06] et les méthodes basées sur des techniques pair-à-pair (P2P) [Abd05, Ali07, Akb07, Che05, Dov03, Fil04, Iam01, Iam04, Jea08, Mar05, Mas05, Pup05, Tal05, Xia05].

Les méthodes basées sur une approche client/serveur peuvent être classées en deux sous-catégories suivant la manière dont les métadonnées sont structurées : centralisée[9] [Dee04, Fit97] ou hiérarchiques[10] [Ant05, Che00, Cza01, Hua04, Jay04, Kau07, Pas08, Ram06, Sch06].

[9] Ces méthodes seront nommées les méthodes centralisées dans la suite du document
[10] Ces méthodes seront nommées les méthodes hiérarchiques dans la suite du document

Les méthodes basées sur des agents peuvent être classées en deux sous-catégories, suivant la manière dont les agents coopèrent avec les composants du système (e.g. agents, ressources) : hiérarchiques [Cao01, Cao02, Fuk06, He05, Yan07, Zhe06] et directs [Din05, Kak06].

Les méthodes basées sur des techniques P2P peuvent être classées en trois sous-catégories, suivant la manière dont les pairs sont organisés : les méthodes P2P non structurées[11] [Abd05, Iam01, Iam04, Jea08, Tal05], les méthodes P2P structurées[12] [Ali07, And02, Che05, Spe03, Cai03, Mis04, Opp03, Tal07b, Xia05] et les méthodes super-pairs[13] [Fil04, Mas05, Mar05, Pup05].

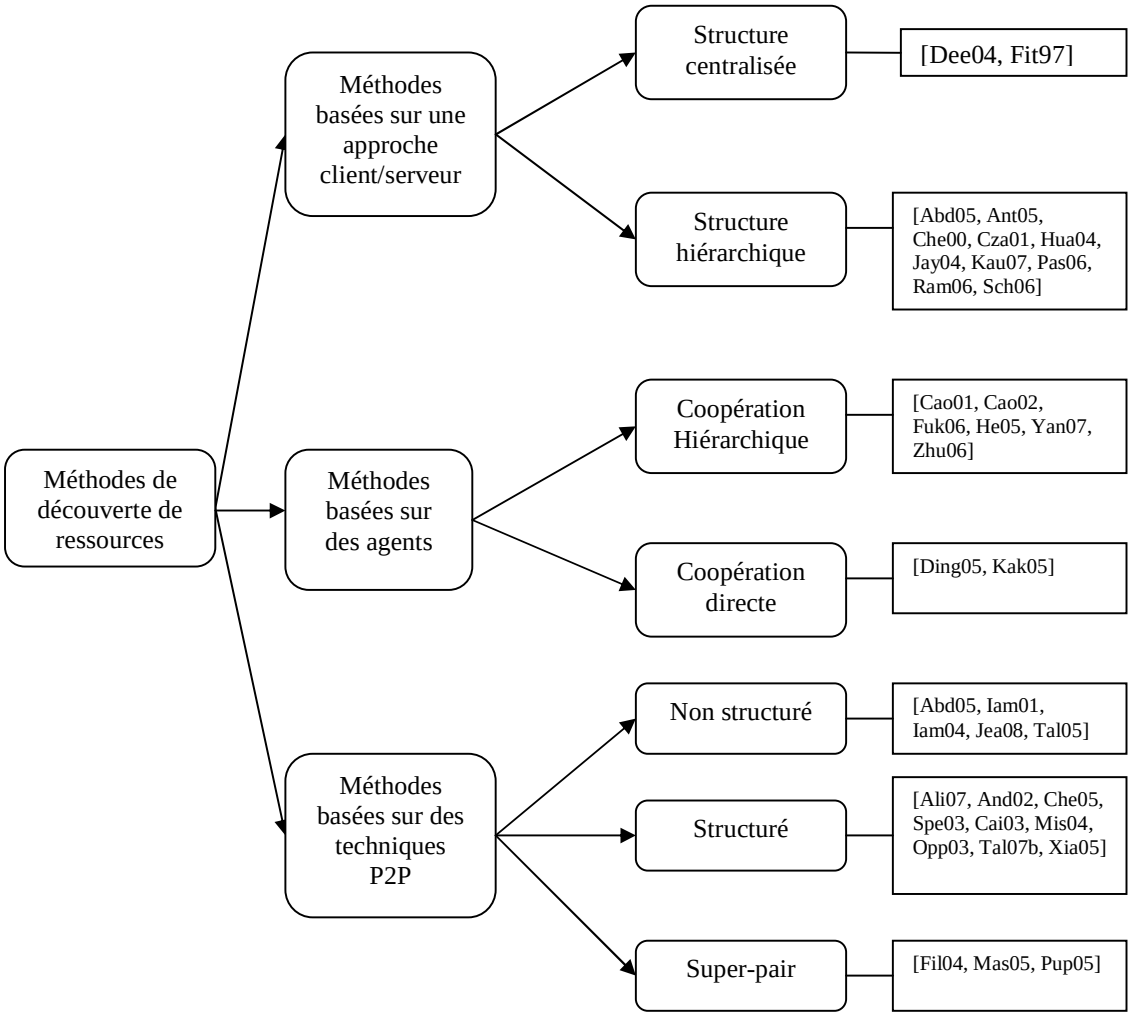

Figure 1 : Classement des méthodes de découverte de ressources

[11] « les méthodes P2P non structurées » désignent « les méthodes basées sur des techniques P2P non structurées »
[12] « les méthodes P2P structurées» désignent « les méthodes basées sur des techniques P2P structurées »
[13] « les méthodes super-pairs » désignent « les méthodes basées sur des techniques super-pairs »

Dans la suite, nous souhaitons étudier d'une manière approfondie puis comparer les méthodes d'une même catégorie. Ensuite, nous effectuons une comparaison globale entre ces trois catégories de méthode suivant un ensemble de critères détaillés dans la section suivante.

2. Critères de comparaison

Afin de pouvoir comparer les méthodes de découverte de ressources, il est nécessaire de présenter l'ensemble des critères considérés comme les plus importants dans un système de grille. Plusieurs études récentes [Ham09, Sad08, Mes08, Ran08, Sed08, Tru07] ont été effectuées pour la comparaison des méthodes de découverte de ressources dans un système de grille. Les trois papiers de [Tru07, Mes08, Ran08] présentent une étude et une comparaison des méthodes de découverte de ressources basées sur des techniques P2P. Plus précisément :

- [Tru07] présente une étude approfondie des systèmes P2P employés aujourd'hui en montrant les avantages et les inconvénients de ceux-ci, et en particulier des systèmes P2P structurés et des systèmes P2P non structurés. Ensuite, les méthodes de découverte de ressources (basées sur ces systèmes) sont présentées, en détaillant les méthodes traitant des requêtes à intervalle (e.g. trouver un ou plusieurs nœuds dont la vitesse CPU est comprise entre 500Mhz et 1000 Mhz) et à multi-attribut (e.g. trouver un ou plusieurs nœuds dont la vitesse CPU = 512 Mhz et dont la RAM = 1 Go) pour des ressources de calcul, puis comparées suivant un ensemble de critères (e.g. protocole utilisé, partage de la charge du travail, traitement des requêtes à multi-attribut).
- [Ran08] présente un tutorial des méthodes de découverte de ressources basées sur des techniques P2P. Plus précisément, [Ran08] présente (i) une taxonomie approfondie sur les ressources d'une grille de calculs, (ii) une taxonomie sur les systèmes P2P, (iii) une étude détaillée sur les méthodes de découverte de ressources de calcul dans un système de grille qui traite les requêtes à intervalle et à multi-attribut et (iv) une comparaison des méthodes citées dans (iii) tout en respectant le passage à l'échelle et la complexité de la découverte en terme de nombre de messages. Pour [Ran08], une requête de découverte de ressources peut être unidimensionnelle (i.e. elle concerne un seul attribut) ou multidimensionnelle (i.e. elle concerne plusieurs attributs). [Ran08] s'intéresse plus particulièrement, à l'étude des travaux

effectués sur l'extension des systèmes P2P structurés pour le traitement de requêtes à multi-attribut dans un système de grille. Une vingtaine de travaux sont détaillés dans ce contexte.

– [Mes08] s'intéresse à l'étude des mécanismes de découverte de ressources pour plusieurs types de réseaux (e.g. les réseaux non-IP comme les réseaux sans fils). Une comparaison entre les mécanismes de découverte est effectuée suivant ces critères : le type du réseau (e.g. filaire, sans fil), l'architecture (e.g. client/serveur, P2P), le passage à l'échelle et la technique de découverte (e.g. structurée, non-structurée).

Suite à l'analyse profonde des travaux actuels sur la découverte de ressources et des travaux de [Mes08, Ran08, Tru07], nous définissons un ensemble de critères cruciaux pour la conception d'une méthode efficace de découverte de ressources, à savoir : le passage à l'échelle, l'instabilité, la propagation de la découverte et de la complexité, la maintenance et la découverte fiable. Pour éviter toute ambiguïté, nous définissons ces critères :

- Le passage à l'échelle : Il précise si une méthode de découverte de ressources peut opérer dans un système constitué d'un très grand nombre de nœuds sans mauvaises performances (e.g. goulet d'étranglement).
- L'instabilité : Elle indique si une méthode de découverte de ressources peut opérer en présence de l'instabilité du système.
- La propagation de la découverte : Elle avertit comment une requête de découverte est propagée (i.e. unicast, broadcast, mutlicast, anycast) dans le système.
- la complexité : Elle précise la complexité d'une méthode de découverte en terme de temps de réponse (coût de traitement local et coût de communication) et du nombre de messages généré.
- Maintenance : Elle renseigne si une méthode de découverte de ressources nécessite un coût important de maintenance surtout en présence de l'effet « churn » (i.e. le processus continu d'entrée et de sortie des nœuds [Rhe04, Wu06]) ou du changement rapide d'état de ressources (e.g. charge CPU, charge E/S). Un coût élevé de maintenance peut provoquer une congestion dans le réseau.
- La découverte fiable : Elle avertit si une méthode de découverte de ressources est toujours capable de répondre à une requête de découverte en précisant si la ressource est bien présente

ou non dans le système (e.g. préciser si une base de données « Patient » est bien connectée au système).

- Les requêtes complexes sont des requêtes qui peuvent porter sur plusieurs attributs nommés requêtes à multi-attribut ou des requêtes à intervalle. Les requêtes complexes peuvent inclure ou non des métadonnées dynamiques (e.g. charge CPU, bande passante, charge RAM). Ce type de traitement concerne surtout les ressources de type calcul.

Il existe d'autres critères (e.g. l'extensibilité, l'adaptabilité [Ran08]) qui peuvent être pris en considération mais nous préférons présenter les critères les plus importants cités ci-dessus. Dans les études ci-dessus, la découverte de ressources est centrée sur la découverte de ressources de type calcul. Comme nous l'avions déjà indiqué dans l'introduction, nous nous intéressons à la découverte de sources de données pour l'évaluation de requêtes en environnement de grille de données. Ainsi, nous précisons la spécificité de la découverte de ressources de calcul par rapport à la découverte de sources de données :

(i) Elle est conçue pour traiter des requêtes complexes (e.g. des requêtes à intervalle) incluant (ou non) des métadonnées dynamiques.

(ii) Elle est souvent approximative (e.g. une machine possédant une vitesse CPU de 1 Ghz peut être utilisée pour une tache nécessitant un CPU de vitesse 2 Ghz).

La découverte de ressources de calcul peut être appliquée pour la découverte de sources de données en ignorant le traitement de requêtes complexes. La découverte approximative n'est pas adaptée aux sources de données car ces dernières sont indispensables pour l'exécution d'une tâche. Ainsi, il faut étendre les méthodes actuelles pour la découverte de ressources de type calcul (ou proposer des nouvelles méthodes) pour qu'elles soient fiables.

Dans la suite, nous présentons le principe d'une manière abstraite de chaque catégorie de méthodes et nous détaillons quelques méthodes proposées dans chaque catégorie. Puis, une discussion est effectuée afin d'analyser les avantages et les limites des méthodes proposées dans chaque sous-catégorie en fonction des critères cités ci-dessus. Enfin, une comparaison globale est effectuée entre les méthodes basées sur une approche client/serveur, les méthodes P2P et les méthodes basées sur des agents dans la section conclusion.

3. Les méthodes basées sur une approche client/serveur.

Dans cette catégorie de méthodes, la découverte de ressources est gérée suivant une approche client/serveur. Dans ce contexte, un serveur a pour rôle de stocker les métadonnées décrivant toutes les ressources du système dans une structure centralisée ou hiérarchique.

Dans les méthodes centralisées [Dee04, Fit97], l'utilisation d'un serveur central n'est pas adaptée à un système de grille car : (i) un serveur central peut rapidement devenir un goulet d'étranglement si un grand nombre de messages est envoyé vers ce serveur et (ii) la panne du serveur central rend le système inutilisable. Ainsi, nous ne détaillons pas cette catégorie de méthodes qui est clairement mal adaptée à un système instable et à grande échelle.

3.1. Les méthodes hiérarchiques

3.1.1. Principe

Dans les méthodes hiérarchiques [Ant05, Che00, Cza01, Hua04, Jay04, Kau07, Pas08, Ram06, Sch06], des serveurs sont interconnectés suivant une hiérarchie prédéfini par les administrateurs (ou l'administrateur) de la grille Informatique. Les nœuds du niveau 0 sont des nœuds clients. A partir du niveau 1, il existe un serveur responsable d'agréger [Cza01, Kau07] et de publier les métadonnées décrivant les ressources (e.g. CPU, E/S, données) d'une partie du système (e.g. une simple organisation ou une OV).

Le serveur du niveau *(n)* agrège les métadonnées d'un groupe de serveur du niveau (*n-1)* jusqu'à arriver au serveur racine. Lorsqu'un serveur reçoit une requête de découverte, elle est évaluée localement puis elle est propagée –en fonction de la présence des métadonnées- aux serveurs du niveau supérieur, suivant un mode unicast, jusqu'à arriver au serveur racine et elle est évaluée localement au niveau de chaque serveur [Mas08] (Figure 2). Une requête de découverte possède alors une complexité en terme de temps de réponse (resp. en terme de nombre de messages) de $O(H+\Sigma C_{i(1 \leq i \leq H)})$ (resp.$O(H)$) où H est le niveau du serveur racine et C_i est le coût nécessaire pour la comparaison d'une requête de découverte avec les métadonnées des ressources d'un serveur.

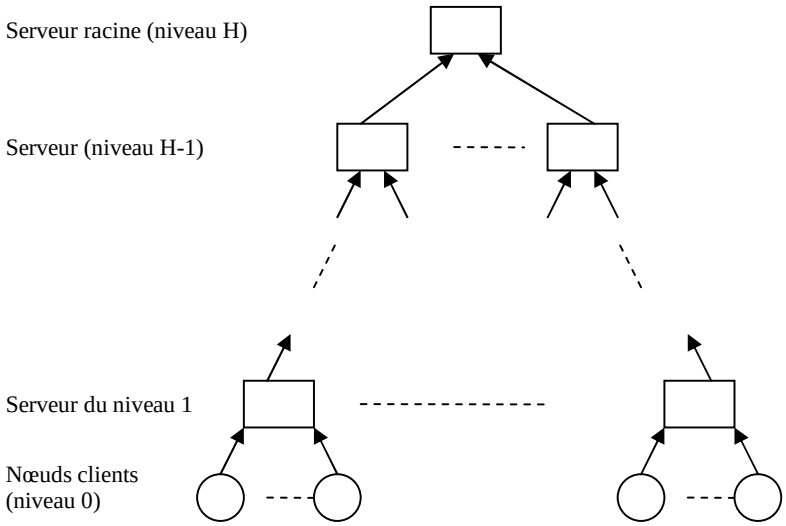

Figure 2 : Topologie hiérarchique [Mas08]

3.1.2. Méthodes

Les méthodes hiérarchiques sont utilisées aujourd'hui dans Globus [Fos05]. Globus est considéré comme l'outil le plus répandu pour le développement des applications tournant sur un système de grille. Dans Globus, les serveurs utilisés sont nommés Index Service (IS). Ces IS sont responsables de la gestion des métadonnées décrivant les ressources d'une OV et sont interconnectés suivant une topologie hiérarchique. Les applications tournant sur Globus sont implémentées en utilisant les services web.

Les services web sont indépendants de toute plate-forme particulière. Ils sont fondés sur quatre technologies : XML (eXtensible Markup Language), SOAP (Simple Object Access Protocol), WSDL (Web Services Description Language) et UDDI (Universal Description, Discovery and Integration). XML est utilisé pour étiqueter les données. SOAP est un protocole permettant la transmission des messages entre des objets distants, indépendamment de la plate-forme installée sur chaque nœud. WSDL sert à décrire les services disponibles et UDDI est utilisé pour lister l'ensemble des services disponibles

Le fruit de la combinaison de la technologie des services web et de la grille informatique a donné naissance à l'Open Grid Services Architecture (OGSA). OGSA a été suggérée par I. Foster dans un papier intitulé « The Physiology of the Grid » [Fos02] puis elle a été développée par le Global Grid Forum (GGF). L'idée est de permettre l'intégration des services et le partage de ressources provenant de plusieurs OVs distribuées à grande échelle. Depuis, il est devenu un

paradigme [Hua04] pour le développement des applications tournant dans un système de grille. Aujourd'hui, la plupart des méthodes hiérarchiques [Ant05, Che00, Hua04, Al-H05, Jay04, Kau07, Lyn09, Ram06, Sch06] sont implémentées sous OGSA. Dans OGSA, l'outil responsable de la découverte de ressources est le Monitoring and Discovery System (MDS). Initialement centralisé, le MDS-1 est passé à une structure hiérarchique [Sch06] dans sa dernière version MDS-4. Dans la suite, nous détaillons comment les services web peuvent être utilisés pour la découverte de ressources dans une topologie hiérarchique.

Méthodes hiérarchiques implémentées avec des services web

Dans les services web classiques chaque serveur est représenté par un certain registre nommé Universal Description Discovery and Integration (UDDI). L'UDDI contient les métadonnées statiques décrivant les ressources disponibles au sein d'une OV. Le problème principal de l'UDDI est que l'administrateur d'une OV (ou d'une partie d'OV) doit manuellement signaler l'arrivée d'une nouvelle ressource [Kau07, Pas08]. Ceci n'est pas pratique surtout lorsque l'ajout de sources de données est assez fréquente [Pac07] (ce qui est bien le cas d'une grille de données) ou si la valeur d'un paramètre d'une ressource peut changer de valeur dans le temps (e.g. valeur de la charge CPU). Dans cette perspective, [Kau07] propose d'enrichir l'UDDI avec un « Rich Query Model » pour la prise en compte de ces contraintes.

Dans MDS-4, l'UDDI est étendu en fournissant 2 services : Index Service (IS) et Trigger Service (TS). Le rôle principal du IS est de publier les métadonnées décrivant les ressources (de type données et calculs) pour permettre leur découverte plus tard. L'IS possède aussi un cache permettant de sauvegarder la dernière information concernant une ressource. Pour maintenir les métadonnées à jour, le TS définit un ensemble de règles pour la mise à jour de celles-ci. Par exemple, si une relation change de nom ou de taille alors il faut le mettre à jour dans le IS (idem si la charge CPU d'un nœud change).

Les étapes de la découverte d'une ressource *R* avec les services web sont les suivantes (la découverte de ressources est implémentée comme un service de grille) [Kau07] :

1. Lors de l'arrivée d'une nouvelle ressource *R*, le GSP (Grid Service Provider) signale l'arrivée de *R* en envoyant un message à l'UDDI avec un fichier GWSDL (Grid Web Service Description Langage). Ce fichier contient une description pour accéder à *R*.

2. Lorsqu'un client veut découvrir *R*, il envoie un message « get » via http (hyper text transfer protocol) à l'UDDI.
3. L'UDDI répond au client en lui envoyant le fichier GWSDL correspondant à *R*.
4. Le client peut désormais envoyer une requête au GSP grâce au ficher GWSDL. La requête est envoyée en utilisant le protocole SOAP.
5. le GSP répond en renvoyant la ressource désirée avec le protocole SOAP.

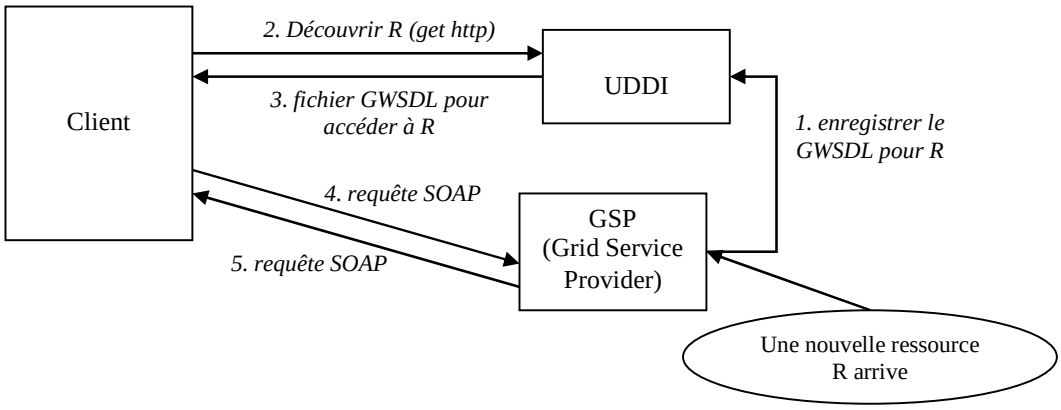

Figure 3 : Étapes de découverte de ressources avec des services Web dans un système de grille [Kau07]

La plupart des méthodes de découverte de sources de données sont implémentés en utilisant les services web comme dans le cadre du projet OGSA-DQP [Alp05, Ant05, Lyn09], du projet DartGrid [Zhu05] mais aussi des travaux de [Jay04]. Le principal avantage de ces méthodes est la garantie d'une découverte fiable. Le second avantage est la compatibilité avec OGSA.

3.1.3. Discussion

Les méthodes hiérarchiques permettent de mieux partager la charge du travail ce qui permet un meilleur passage à l'échelle en comparaison avec les méthodes centralisées. Toutefois, un serveur peut devenir un goulet d'étranglement si un grand nombre de requêtes de découverte est envoyé vers ce serveur. La panne d'un serveur peut rendre inaccessible les métadonnées décrivant les ressources d'une partie du système (e.g. une OV). Pire, la panne du serveur racine rend impossible la communication entre un nombre d'OVs dans le système. Donc, ces méthodes ne sont pas adaptées à l'instabilité du système.

La propagation de la découverte est faite suivant une succession d'unicast. Une requête de découverte possède alors une complexité en terme de temps de réponse (resp. en terme de nombre de messages) de $O(H+\Sigma C_{i(\,1 \le i \le H)})$ (resp.$O(H)$) où H est le niveau du serveur racine et C_i est le coût du traitement local au niveau du serveur i. Ci peut devenir important si un serveur stocke un grand nombre de métadonnées (c'est surtout le cas du serveur racine).

La maintenance du système est plus compliquée par rapport à un système centralisé car cela nécessite un nombre important de messages de communication entre les serveurs organisés en hiérarchie, surtout en présence de l'effet churn. En pratique, un serveur (ou un IS) ne peut pas gérer toutes les ressources venant des serveurs descendants (du niveau moins bas) pour les raisons suivantes [Mas08] : la difficulté de la gestion des métadonnées statiques concernant des sources de données (dont l'ajout est très fréquent [Pac07]) car cela nécessite une mise à jour très coûteuse dans la hiérarchie et (ii) la limite en terme de mémoire, CPU et E/S pour la gestion d'un nombre très grand de ressources (qui peut atteindre des millions).

La découverte est fiable vu que tous les serveurs peuvent être contactés depuis le lancement du processus la découverte d'une ressource, jusqu'à l'arrivée au dernier serveur (la racine).

Ces méthodes peuvent traiter des requêtes complexes incluant des métadonnées dynamiques (e.g. [Kau07, Sch06]).

Ces méthodes sont mal adaptées face à la dynamicité des nœuds car un serveur peut toujours tombé en panne. Dans cette perspective, un groupe nommé OGSA-P2P a été créé pour la combinaison des systèmes P2P (qui peuvent opérés dans un système instable) avec OGSA [Bha05].

4. Les méthodes basées sur des agents.

Un système de grille peut être vu comme une infrastructure offrant un ensemble de services permettant de partager les ressources de plusieurs OVs instables et à grande échelle. Un tel système nécessite souvent la prise de décisions d'une manière flexible et autonome [Fos04]. Les agents qui sont, par définition, autonomes et intelligents peuvent être très utiles dans un tel système (e.g. si un agent constate la déconnexion d'un nœud, il peut l'indiquer au service d'information du système). En effet, les agents mobiles permettent de réduire le coût de la communication lors de la découverte de ressources en évitant les interactions distantes (e.g. avec

un serveur distant) par des interactions locales (e.g. entre un agent et la ressource elle-même [Yan07, Yu06]).

Dans ce contexte, plusieurs travaux se sont intéressés à l'utilisation de la technologie des agents pour la découverte de ressources dans un système de grille. L'idée principale de l'utilisation des agents est de permettre la coopération et la négociation [Din05, Fos04, Kak06, Yu06, Yan07] des agents avec d'autres composants du système (e.g. d'autres agents, une ressource, un serveur). Les méthodes de découverte de ressources basées sur des agents peuvent être classées en 2 catégories suivant la manière dont les agents coopèrent avec les composants du système : hiérarchique [Cao01, Cao02, He05, Zhu06, Yan07] et directe [Kak06, Din05].

4.1. Coopération hiérarchique

4.1.1. Principe

Dans ces méthodes, les agents sont utilisés pour aider à mieux découvrir les ressources suivant un mode de coopération hiérarchique. Les agents, organisés en hiérarchie, peuvent : stocker eux-mêmes les métadonnées décrivant les ressources [Cao01, Cao02, He05, Zhu06], ou contacter le service d'information d'un système de grille (e.g. une hiérarchie de serveurs LDAP [Yan07]).

4.1.2. Méthodes

[Cao01, Cao02, He05] proposent une hiérarchie d'agents identiques (i.e. ils ont les mêmes fonctions), où chaque agent est capable de coopérer avec d'autres agents, pour la découverte de ressources dans un système de grille. La Figure 4 montre le modèle hiérarchique d'agent composé de 3 niveaux [He05] : le « Broker » (i.e. le courtier) est l'agent principal responsable de toute la hiérarchie qui maintient les métadonnées statiques de toutes les ressources (e.g. service) du système, l'agent « lookup » permet la recherche des ressources dans sa sous-hiérarchie et l'agent du niveau le plus bas (la feuille) maintient l'information sur un ensemble de ressources. Chaque agent dans le système possède un et un seul supérieur. Un agent peut gérer une sous-hiérarchie d'agents [Cao01, He05].

La coopération entre un agent *A* et d'autres agents se fait lors de [Cao01, He05] : (i) la publication des métadonnées statiques décrivant les ressources dont *A* est responsable dans

l'hiérarchie et (ii) la découverte de ces métadonnées en contactant son supérieur (ou le supérieur de son supérieur et ainsi de suite).

La publication et la découverte sont effectuées en fonction d'une certaine table associée à chaque agent (e.g. table d'information d'agent *AIT* dans [He05] et table de capacité d'agent [Cao01, Cao02]). L'*AIT* d'un agent *A* est constituée de 3 parties (Figure 4) [He05]:

- *UP_AIT* permet de stocker des métadonnées sur l'agent supérieur à *A* (e.g. adresse),
- *H_ AIT* permet de stocker toutes les informations (e.g. métadonnées décrivant un ensemble de ressources) que possèdent l'agent *A* et des informations sur les agents du niveau plus bas,
- *V_ AIT* permet de stocker l'information sur les agents mobiles passagers (du niveau plus bas) qui ont contactés *A* temporairement pendant leur mouvement.

Lorsqu'une requête de découverte d'une ressource *R* est soumise à un agent *A*, *A* cherche d'abord s'il possède l'adresse de l'agent responsable de *R* via son *H_AIT*. Si c'est le cas, il redirige la requête directement vers l'agent responsable de *R*. Sinon, il contacte son supérieur via son *UP_AIT* suivant le même principe (puis le supérieur de son supérieur et ainsi de suite).

Ainsi, la découverte est propagée suivant un mode unicast et elle est fiable. Une requête de découverte possède alors une complexité en terme de temps de réponse (resp. en terme de nombre de messages) de $O(H+\Sigma C_{i(1 \le i \le H)})$ (resp.$O(H)$) où *H* est le niveau de l'agent racine (le broker) et C_i est le coût du traitement local pour un agent *i* (e.g. coût de comparaison, coopération entre l'agent du niveau *i* et du niveau *i+1*).

Le modèle présenté dans [Cao01, Cao02, He05] permet des opérations simples, i.e. pour une ressource donnée, trouver l'agent responsable de cette ressource. Ceci peut être facilement étendu pour traiter des requêtes complexes (e.g. à multi-attribut) car aucune fonction de hachage n'est effectuée sur le nom de la ressource à découvrir. La découverte des métadonnées dynamiques n'est pas traitée dans [Cao01, He05].

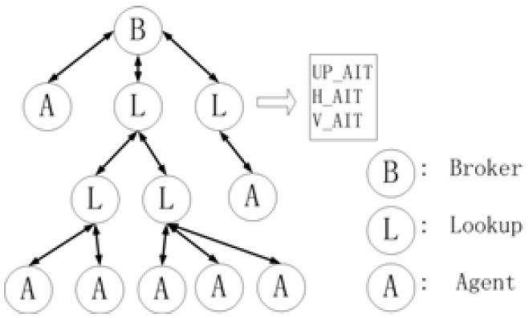

Figure 4 : Modèle hiérarchique d'agents [He05]

[Yan07] propose un mécanisme de découverte de ressources basé sur une coopération hiérarchique des agents mobiles avec des serveurs d'un système de grille. L'idée est d'utiliser les propriétés des agents (e.g. autonomie, intelligence) pour réduire le coût de la découverte de ressources et pour faciliter la mise à jour des métadonnées décrivant les ressources. Si un agent constate la panne d'un nœud lors de son déplacement, il peut l'indiquer au serveur responsable de cette ressource. D'autre part, un agent peut aider dans la maintenance des métadonnées dynamiques. En effet, un agent peut capturer le changement d'état de ressources (e.g. charge CPU) dans une OV et l'indiquer au serveur de cette OV.

Dans [Yan07], le système est supposé être constituée de plusieurs OVs où chaque OV est composée de plusieurs nœuds (e.g. une machine parallèle, une grappe de machines). Trois niveaux de serveurs LDAP (Light Data Access Protocol) existent : un serveur LDAP est associé à chaque nœud, un serveur LDAP est associé à chaque OV et un serveur LDAP global permet l'interconnexion entre toutes les OVs.

Lorsqu'une requête de découverte d'une ressource *R* est soumise à un nœud local, le système vérifie d'abord si les métadonnées décrivant *R* sont stockées dans le serveur LDAP du nœud local. Si c'est le cas, il envoie directement les métadonnées de *R* au nœud qui a initié la découverte. Dans le cas contraire, un agent mobile *AM* contacte le serveur LDAP de l'OV local qui, à son tour, vérifie dans sa base si *R* appartient à son OV suivant le même principe. Si *R* n'appartient pas à l'OV local, *AM* se déplace au niveau du serveur LDAP global pour la découverte de *R*. Le serveur LDAP global envoie, en parallèle, plusieurs agents mobiles aux OVs voisines [Yan07].

L'utilisation des agents mobiles peut aider à réduire le goulet d'étranglement au niveau des serveurs centraux (e.g. un agent peut prendre une décision d'une manière autonome sans

consulter le serveur d'information). Par contre, comme les métadonnées décrivant les ressources sont stockées dans des serveurs LDAP, la panne d'un serveur peut entrainer le dysfonctionnement d'une partie du système (e.g. une OV) voir le dysfonctionnement de tout le système. Ainsi, la méthode de [Yan07] est mal adaptée à l'instabilité du système.

La découverte est propagée suivant une succession d'unicast et elle est fiable. Une requête de découverte possède une complexité en terme de temps de réponse (resp. en terme de nombre de messages) de $O(H+\Sigma C_{i(1 \leq i \leq H)})$ (resp.$O(H+M)$) où H est le niveau du serveur global, C_i est le coût du traitement local pour un agent i (e.g. coût de comparaison, coopération entre un agent et le serveur du niveau i) et M est le nombre de messages de découverte envoyé par le serveur LDAP Global aux OVs voisines.

[Yan07] peut traiter des requêtes complexes incluant des métadonnées dynamiques car un agent s'adresse directement à un serveur lors de la découverte de ressources. Un serveur peut facilement implémenter les techniques nécessaires pour le traitement de requêtes complexes.

4.1.3. Discussion

Dans les travaux de [Cao01, Cao02, He05], un agent peut stocker les métadonnées statiques décrivant les ressources et ne pas se référer à un nœud ou à un serveur central dans le système. Cela permet un meilleur passage à l'échelle comparé à [Yan07] où ces métadonnées sont récupérées par un agent en contactant un serveur LDAP.

Dans [Yan07], la panne d'un serveur peut entraîner le dysfonctionnement d'une partie du système à la différence de [Cao01, Cao02, He05] où aucun nœud central n'est utilisé. Par contre, la taille d'un agent [Cao01, Cao02, He05] peut devenir très grande vu qu'il doit maintenir les métadonnées sur une partie des ressources d'un système à grande échelle (l'agent du niveau le plus supérieur doit maintenir des métadonnées sur un très grand nombre de ressources). Le déplacement des agents peut devenir très couteux. Ainsi, les méthodes basées sur ceux-ci en mode coopération hiérarchique ne sont pas adaptées à un système instable et à grande échelle.

La propagation de la découverte est effectuée en mode unicast hiérarchique. Une requête de découverte possède une complexité en terme de temps de réponse (resp. en terme de nombre de messages) qui est proportionnelle à la hauteur H du serveur racine (ou global) [Yan07] ou de l'agent racine [Cao02, Din05] plus le coût du traitement local (resp. proportionnelle à la hauteur

H du serveur racine [Yan07] ou de l'agent racine [Cao02, Din05]). Le coût du traitement local peut être, par exemple, le coût de la coopération entre deux agents.

Dans [Yan07], l'emploi des agents mobiles peut aider à réduire le coût de maintenance du système en indiquant le départ d'un nœud ou le changement d'état de ressources. Dans [Cao01, Cao02, He05], la maintenance de l'agent le plus haut placé dans la hiérarchie est très difficile et la profondeur de celle-ci doit être la plus petite possible. D'autre part, la communication entre les agents peut poser un problème à cause de l'effet churn car ce dernier peut nécessiter un nombre important de messages afin de maintenir les liens de communications entre les agents.

La découverte est fiable car un agent contacte tous les serveurs (ou tous ses agents supérieurs) jusqu'à arriver au serveur racine lors d'une découverte. Les travaux de [Yan07, Cao01, Cao02, He05] peuvent traiter des requêtes complexes. [Yan07] traite des requêtes incluant des métadonnées dynamiques alors que dans [Cao01, Cao02, He05] ceci n'est pas indiqué.

4.2. Coopération directe.

4.2.1. Principe

Dans ces méthodes, les agents sont utilisés afin de mieux découvrir les ressources suivant un mode de coopération directe. [Kak06, Din05] présentent une technique permettant la publication et la découverte de ressources dans un système de grille. [Din05] précise que l'organisation des agents en hiérarchie n'est pas efficace surtout si l'ajout des ressources est assez fréquent car cela entraîne une mise à jour, concernant une sous-hiérarchie d'agents et qui peut être très coûteuse dans un système à grande échelle.

4.2.2. Méthodes

[Din05] nous montre une technique permettant la publication et la découverte de ressources dans un système de grille. Le principe est d'organiser les agents dans un graphe sans aucune hiérarchie. Chaque agent stocke les métadonnées statiques et dynamiques décrivant les ressources dans des tables nommées tables d'information d'agents (Agent Information Tables AIT). Un agent *A* possède deux types d'*AIT* : une *T_AIT* qui stocke les métadonnées concernant les ressources gérées par *A* et une *A_AIT* qui stocke les métadonnées concernant des ressources, reçues par des agents adjacents (ou voisins).

Lorsqu'une nouvelle ressource *R* arrive dans le système, l'agent responsable de cette ressource met à jour sa table *T_AIT* et informe ses agents adjacents de l'arrivée de *Rnew*.

La découverte de ressources revient à la recherche des métadonnées dans les *AITs*. Lorsqu'un agent *A* reçoit une requête de découverte de ressource *R*, il recherche d'abord dans son *T_AIT*. Si la découverte réussit, *A* communique les métadonnées concernant *R*, sinon il recherche dans son *A_AIT* si un de ses agents adjacents possède les métadonnées décrivant *R*. Si c'est le cas, la requête de découverte de *R* est envoyée directement à l'agent possédant les métadonnées de *R*. Sinon, *A* envoie la requête de découverte à un de ses agents adjacents *Adj_A* qui à son tour vérifie ses *AITs* (d'abord *T_AIT* puis *A_AIT*). Cela est répété en contactant un adjacent de *Adj_A* jusqu'à trouver les métadonnées décrivant *R*. La recherche n'est pas limitée par un Time To Live (TTL), (qui permet de définir un nombre de sauts pour la propagation d'une requête). Ainsi, elle est exécutée jusqu'à ne plus trouver un agent à contacter (i.e. recherche en boucle).

La découverte est propagée suivant une succession d'unicast, ce qui permet de passer à l'échelle car aucun mécanisme de diffusion n'est employé. Cependant, la découverte n'est pas fiable. En plus, la méthode peut opérer en présence de l'instabilité des nœuds car un agent peut toujours contacter un agent adjacent, sauf dans le cas où tous ceux-là tombent en panne. La complexité de la découverte, en terme de temps de réponse (resp. en terme de nombre de messages), est de *O(d)* (resp. *O(d*) où *d* est le diamètre de graphe des agents). Le coût de traitement local est négligeable devant le coût de communication car tous les agents se partagent la charge du travail. La méthode peut traiter les requêtes complexes incluant des métadonnées dynamiques.

[Kak06] présente une méthode, basée sur des agents, pour la découverte de ressources. Dans cette perspective, trois types d'agents coopèrent ensemble : un agent local (L), un agent représentant un ensemble de ressources (RR) et un agent client (C). Le rôle de chaque agent sera détaillé dans la suite :

- Un agent L représente un nœud du système (e.g. une machine parallèle). L communique ses métadonnées statiques (e.g. nombre de processeurs) à un ou plusieurs RRs pour que les ressources de L soit visible au niveau du système.
- Un agent RR contient un registre responsable de stocker les métadonnées statiques décrivant un ensemble de ressources et il est connecté à plusieurs RRs voisins. Lorsqu'un RR reçoit une requête de découverte, il vérifie d'abord son registre local. Si le RR ne peut pas répondre à la

requête, il redirige cette requête à un ensemble de voisins RR selon un mode anycast. Le nombre de requêtes de découverte est limité par un TTL.

– Un agent C émet une requête de découverte à un RR. Si C ne reçoit pas une réponse pendant une certaine période, la requête est transférée à un ou plusieurs RRs (mode unicast). Lorsque C estime que le nombre de ressources découvertes est suffisant, il négocie directement avec des agents L responsables de ces ressources pour la sélection de ressources. Un agent L peut donner l'agrément ou non à C en fonction de la charge actuelle de la ressource dont il est responsable.

La méthode peut passer à l'échelle car la découverte n'est pas effectuée par diffusion et il n'existe pas un nœud central dans le système (les RRs doivent être bien distribués dans le système). Si un RR tombe en panne, ceci peut entraîner l'inaccessibilité d'une partie du système. Les RRs doivent être maintenus lors de l'ajout de nouvelles ressources mais la mise à jour est consacrée uniquement aux métadonnées statiques. La découverte est effectuée par un mode anycast (car uniquement un sous-ensemble de RR est choisi).

La complexité de la découverte en terme de temps de réponse (resp. en terme de nombre de messages) est $O(d+\Sigma C_{i(1 \leq i \leq d)})$ où d est le diamètre du réseau RRs et Ci le coût du traitement local au niveau de chaque RR (resp. $O(d)$ où d est le diamètre du réseau RRs).

Enfin, la méthode [Kak06] peut traiter des requêtes complexes incluant des métadonnées dynamiques. Ces dernières sont récupérées pendant la phase de sélection de ressources.

4.2.3. Discussion

Les méthodes basées sur des agents mobiles en mode coopération directe permettent un passage à l'échelle dans un système instable [Din05]. Par contre, l'utilisation des registres (i.e. des serveurs) [Kak06] contenant des métadonnées décrivant les ressources peuvent former un goulet d'étranglement.

La découverte est propagée suivant une succession d'unicast [Din05] ou anycast [Kak06]. Ainsi, la découverte n'est pas fiable. La complexité de la découverte en terme de temps de réponse (resp. en terme de nombre de messages) est proportionnelle au diamètre de graphe d'agents [Din05, Kak06] ainsi que le coût du traitement local [Kak06] (resp. proportionnelle au

diamètre de graphe d'agents [Din05, Kak06]). Le coût de traitement local dans un registre dépend de l'ensemble de métadonnées stockées dans ce registre.

Les méthodes [Din05, Kak06] ne peuvent pas garantir une découverte fiable. Pour la garantir, il faut propager une requête de découverte à tous les nœuds du système ce qui peut provoquer une inondation dans un système à grande échelle.

Enfin, les méthodes [Din05, Kak06] peuvent traiter des requêtes complexes incluant des métadonnées dynamiques mais le résultat envoyé est partiel.

5. Les méthodes P2P

L'envie de partager des fichiers (e.g. données, musique, vidéos) à une échelle mondiale comme Internet a donné naissance à de nouveaux systèmes nommés les systèmes P2P. Les premiers systèmes P2P sont apparus au début du XXI^{ème} siècle (e.g. Kazaa et Gnutella). Le service principal offert dans un système P2P est la localisation efficace d'une clé (e.g. le nom d'un fichier) dans un système constitué d'un très grand nombre de nœuds. Les ressources considérées dans un tel système sont souvent des fichiers de données (e.g. musique, vidéos).

Le succès des systèmes P2P est dû aux « bonnes propriétés » de ces systèmes (e.g. la dynamicité, le passage à l'échelle, l'autonomie). Le paradigme pair-à-pair (P2P) a été largement utilisé dans un système de grille [Pac07, Tru07]. Particulièrement les techniques P2P ont été souvent employés pour la découverte de ressources dans un système de grille [Abd05, Iam01, Iam04, Jea08, Tal05, Ali07, And02, Che05, Spe03, Cai03, Sal07, Xia05, Fil04, Mas05, Pup05, Tal05]. Ces méthodes peuvent être classées en trois sous-catégories suivant la manière dont les pairs sont organisés : les méthodes P2P non structurées [Abd05, Iam01, Iam04, Jea08, Tal05], les méthodes P2P structurées [Ali07, And02, Che05, Spe03, Cai03, Sal07, Xia05] et les méthodes super-pairs [Fil04, Mas05, Pup05, Tal05].

5.1. Les méthodes P2P non structurées

5.1.1. Principe

Les méthodes P2P non structurées sont les premières méthodes P2P utilisées dans un système de grille [Iam01, Iam04]. Dans ces méthodes, chaque nœud du système stocke les métadonnées décrivant les ressources locales dans un catalogue local. Chaque nœud est connecté à un ensemble de voisins. Il n'y a ni répertoire centralisé ni contrôle précis sur la topologie du réseau.

Lorsqu'une requête de découverte est soumise à un nœud, un message sera propagé selon un mode de broadcast (i.e. diffusion) à tous les voisins (dans le réseau P2P). Si un de ces voisins possède l'information sur la ressource en question, il répond au pair, qui a lancé la découverte, en lui envoyant les métadonnées décrivant la ressource. Si le pair qui a reçu la requête ne possède pas ces métadonnées alors il propagera la requête à tous ses voisins (sauf l'initiateur). La complexité de la découverte en terme de temps de réponse (resp. en terme de messages) est de $O(d)$ où d est le diamètre du réseau P2P non structuré (resp. $O(N^2)$ messages [Dov03, Tru07] où N est le nombre de nœuds dans le système). On peut supposer que d vaut $O(log(N))$ [Tru07]. Le coût de traitement local est négligeable devant le coût de communication [Mas08] car chaque nœud stocke un petit ensemble de métadonnées (i.e. les métadonnées décrivant les ressources locales). Clairement, la découverte par diffusion peut créer une inondation dans un système constitué d'un très grand nombre de nœuds. Dans ce cas, la découverte est fiable car tous les nœuds sont contactés.

5.1.2. Méthodes

Afin de limiter le nombre de messages propagés lors de la découverte de ressources, Iamnitchi et Foster [Iam01, Iam04] proposent d'utiliser un TTL. Ce choix permettra d'éviter les recherches en boucle mais ne garantit pas des recherches fiables. D'autre part, le choix du TTL n'est pas facile à déterminer. Il dépend de la topologie du réseau. Si le TTL est grand, cela peut surcharger le réseau inutilement. Si le TTL est petit, la recherche d'une ressource peut échouer même si la ressource est bien connectée au système. La complexité de la découverte, lors de l'utilisation d'un mécanisme TTL, en terme de temps de réponse (resp. en terme de messages) est de $O(TTL)$ [Tru07] (resp. $O(n)$ messages [Dov03]) avec TTL < diamètre du réseau P2P non structuré (sinon l'utilisation d'un TLL n'a pas de sens) et $n<N$ (n est un sous-ensemble de nœud du système).

L'utilisation d'un TTL permet de limiter l'espace de recherche car les messages sont uniquement inondés chez les voisins (et chez les voisins des voisins et ainsi de suite jusqu'à l'expiration du TTL). La diffusion de messages de découverte peut limiter le passage à l'échelle du système à un nombre limité de nœuds. Ainsi des stratégies basées sur un mode de propagation unicast sont proposées. Dans ce contexte, [Iam04, Jea08] définissent des stratégies pour la redirection des messages de découverte vers un et un seul voisin. Les stratégies définies sont

basées sur l'utilisation : (i) des protocoles de marches aléatoires [Jea08] et (ii) des caches [Iam04, Jea08], dans un but de réduire la complexité de la découverte.

Les protocoles de marches aléatoires consistent à envoyer une requête à un seul voisin, choisi aléatoirement, et non à tous les voisins [Jea08]. La profondeur d'une démarche aléatoire peut être limitée par un TTL et l'émetteur d'une requête peut initier plusieurs marches aléatoires jusqu'à trouver une ressource.

L'utilisation des caches [Iam04] consiste à sauvegarder les résultats précédents issus d'une requête de découverte afin de les utiliser plus tard. La maintenance des caches est très dure car l'état des ressources change assez souvent dans une grille et les métadonnées décrivant les ressources doivent être mises à jour en continu. Dans ce contexte, [Jea08] propose une méthode qui améliore la découverte classique basée sur des caches [Iam04] en permettant la coopération entre les caches. Cette méthode nécessite une politique de mise à jour des caches sinon les métadonnées risquent d'être obsolètes.

Enfin, [Jea08] offre un mécanisme pour la découverte des ressources les plus « rares » dans un système de grille (e.g. un calculateur possédant une quantité de mémoire largement supérieure à la norme). Ce mécanisme peut provoquer un surcoût car il nécessite un nombre important de messages.

Lors de l'utilisation de stratégies citées ci-dessus, la complexité de la découverte en terme de temps de réponse (resp. en terme de messages) est de $O(d)$ [Tru07] (resp. $O(d)$ messages) où $O(d)$ est le diamètre du réseau P2P non structuré. Certes, l'utilisation de ces stratégies permet un passage à l'échelle (pas de goulet d'étranglement) mais elle possède des limitations majeures : la découverte n'est pas fiable (e.g. les protocoles de marches aléatoires ne garantissent pas des découvertes fiables) et l'utilisation des caches peut avoir des limites dans un système instable et à grande échelle car les métadonnées décrivant les ressources doivent être mises à jour en continu.

5.1.3. Discussion

Les méthodes non structurées permettent de partager la charge du travail entre tous les nœuds du système vu qu'il n'y a aucun contrôle centralisé dans le système. Ces méthodes sont bien adaptées face à l'instabilité du système vu que tous les nœuds possèdent les mêmes responsabilités. La propagation de la découverte doit être effectuée par diffusion pour permettre

une découverte fiable. La diffusion peut rapidement créer une inondation dans le réseau et empêcher le passage à l'échelle.

La complexité de la découverte en terme de temps de réponse (resp. en terme de messages) est de *O(d)* [Tru07] où *d* le diamètre du réseau P2P non structuré (resp. *O(N)* messages [Dov03, Tru07] où *N* est le nombre de nœuds dans le système). L'utilisation des mécanismes TTL et des stratégies permettent de limiter le nombre de messages propagés mais la découverte n'est pas fiable dans ce cas.

Un système non structuré ne nécessite pas de maintenance en présence de la dynamicité des nœuds ou du changement d'état de ressources sauf si le système emploie une politique de gestion des caches nécessitant une mise à jour de ceux-ci entre les nœuds [Jea08].

Les méthodes non structurées peuvent traiter des requêtes de découverte de ressources complexes. L'utilisation des stratégies (e.g. TTL) peut conduire à des résultats partiels. Pour garantir une découverte fiable, il faut contacter tous les nœuds du système ce qui peut provoquer une inondation dans le réseau.

Les requêtes de découverte peuvent inclure des métadonnées dynamiques. En effet, une requête de découverte contacte directement le nœud responsable d'une ressource *R*. Les métadonnées dynamiques découvertes ne sont pas forcément à jour (e.g. si le nœud contacté est un serveur MDS-4 d'une OV ou autre, alors il n'y a aucune garantie si les métadonnées sont obsolètes ou périmées).

5.2. Les méthodes P2P structurées.

5.2.1. Principe

Les méthodes P2P structurées [Ali07, And02, Che05, Spe03, Cai03, Xia05] utilisent une structure virtuelle pour interconnecter les pairs (ou les nœuds). Cette structure de données est basée sur l'utilisation des Tables de Hachage Distribuées (THD). Une THD est un tableau permettant le stockage et l'accès rapide de paires de type <*clé, valeur*>. Les THDs proposent des opérations basiques comme : *Store(clé, valeur)* (qui permet d'associer une *valeur* pour *clé* dans la THD) et *Lookup(clé)* (retournant la *valeur* associée à *clé* dans la THD). Les THDs améliorent sensiblement l'efficacité d'une table de hachage normale car elles rendent possibles l'exécution de plusieurs opérations en parallèle. Dans ce contexte, nous citons les 3 systèmes P2P structurés Chord [Sto01] (c'est le système P2P structuré le plus employé aujourd'hui), Pastry [Row01]

(développé par Microsoft) et CAN [Rat01]. D'une manière abstraite, le service offert par ces systèmes est le suivant : étant donné une clé, le système renvoie l'adresse du nœud responsable de cette clé en un nombre limité de sauts, plus précisément de complexité *O(log(N))* (en terme de temps de réponse et de messages) avec *N* le nombre de nœuds dans le système [Sto01, Row01]. Le système doit être maintenu pour chaque entrée ou sortie d'un nœud. Le coût de cette mise à jour est de $O(log^2(N))$ messages lors de l'entrée ou de la sortie d'un nœud dans Chord [Sto01].

Les THDs peuvent être facilement appliquées pour la découverte de sources de données. Par contre, la découverte de ressources de type calcul nécessite souvent des découvertes plus complexes qu'une simple recherche (e.g. requête à intervalle). Ce type de découverte est beaucoup plus compliqué qu'une simple recherche dans les THDs car une fonction de hachage est appliquée pour les métadonnées à découvrir. Nous allons détailler comment les techniques P2P structurées (basées sur des THDs) peuvent être étendues pour le traitement de requêtes complexes.

5.2.2. Méthodes pour le traitement de requêtes complexes

Les THDs ont été conçues pour les systèmes P2P supportant ainsi des opérations simples (e.g. découverte d'un fichier). En effet, les THDs utilisent le hachage pour la distribution uniforme des clés et ne peut donc pas supporter des propriétés structurelles [Ram04]sur l'espace des clés (e.g. définir un ordre sur les clés). Or, dans un système de grille, on doit permettre des découvertes plus complexes. Par exemple, une requête peut demander des machines ayant une RAM libre à 0% (Requêtes de découverte de ressources dynamiques). Une requête peut aussi demander un ensemble de CPUs comprises entre 1 et 2 Ghz (requêtes à intervalle) ou aussi un nœud avec un CPU>1Ghz et une RAM de 2 Ghz (requêtes à multi-attribut). Par conséquent, les THDs ont été étendues dans un système de grille afin de traiter des requêtes complexes. Un nombre important de travaux a été proposé pour permettre aux THDs de traiter des requêtes plus complexes dans un système de grille. Nous citons d'abord les requêtes à intervalle [Gao04, Ram04] puis les requêtes à intervalle et à multi-attribut [Che05, And02, Cai03, Bha05, Spe03, Sch03, Opp04]. Les requêtes complexes peuvent inclure (ou non) des métadonnées dynamiques comme dans [And01, Che05].

Méthodes pour le traitement de requêtes à intervalle :

Dans les travaux de [Gao04, Ram04], le problème traité est : comment permettre aux THDs la découverte des clés dans un intervalle donné ? E.g., trouver toutes les clés comprises entre *X* et *Y*, plus précisément l'ensemble de toutes les clés *C* tel que $X \leq C \leq Y$). En effet, les clés ne sont pas ordonnées dans les THDs car ces dernières permettent une distribution uniforme. [Gao04, Ram04] proposent l'utilisation des arbres (i.e. arbre de recherche d'intervalle [Gao04] et arbre de hachage de préfixe [Ram04]) pour le traitement de requêtes à intervalle. D'un point de vue conceptuel, les deux travaux se ressemblent avec, toutefois, une différence : dans [Ram04] seules les feuilles stockent des métadonnées tandis que dans [Gao04] n'importe quel nœud peut stocker des métadonnées. Dans la suite, nous détaillerons les travaux de [Ram04] étant donnée que les deux travaux [Gao04, Ram04] ont des points communs.

[Ram04] propose une structure de données distribuées, nommée arbre de hachage par préfixe PHT. Le PHT peut tourner sur n'importe quel type de THD. Le PHT est un arbre binaire où chaque nœud est étiqueté avec un préfixe défini récursivement (pour un nœud *l*, l'enfant gauche et droit seront étiquetés *l0* et *l1* respectivement). Une requête à intervalle, e.g. $L \leq H$, doit retourner toutes les clés dans les PHT vérifiant $L \leq K \leq H$. Deux algorithmes sont proposés. Le premier consiste d'abord à découvrir la *feuille(L)* en utilisant les opérations de recherche de PHT (basées sur des THDs). Puis, il s'agit de découvrir la *feuille(H)* à partir de *L* d'une manière séquentielle. La complexité de la découverte en terme de temps de réponse (resp. en terme de nombre de messages) est de *O(Log(D)+M)* où *D* le nombre de bits dans *feuille(L)* et *M* le nombre de sauts pour atteindre *feuille(H)* (resp. avec *M* le nombre de messages envoyés pour atteindre la *feuille(H)*).

Le deuxième algorithme proposé est une amélioration (en terme de temps de réponse et non du nombre de messages) du premier. Il s'agit dans un premier temps de découvrir, en utilisant les opérations de PHT, le préfixe commun de *L* et *H*. Ensuite, des requêtes de découverte sont propagées pour trouver le nœud *L* et le nœud *H* indépendamment. La complexité de la découverte en terme de temps de réponse (resp. en terme de nombre de messages) est de *O(Log(P)+max(ML,MH))* (resp. *O(Log(P)+ML+MH*) avec *P* le nombre de bits dans le préfixe en commun pour *L* et *H*, *ML* est le coût pour atteindre *feuille(L)* et *MH* est le coût pour atteindre

feuille(H) (resp. avec *ML* le nombre de messages envoyés pour atteindre la *feuille(L)* et *MH* le nombre de messages envoyés pour atteindre la *feuille(H)*).

Méthodes pour le traitement de requêtes à intervalle à multi-attribut

Plusieurs travaux sont proposés pour le traitement de requêtes à intervalle et à multi-attribut [And02, Che05, Cai03, Bha05, Spe03, Sch03, Opp04, Rat01, Tal07b]. Dans la plupart de ces méthodes, une requête de découverte est décomposée en plusieurs sous-requêtes puis chaque sous-requête est évaluée séparément. Le résultat est rassemblé (par intersection) au niveau du nœud qui a initié la découverte. Par la suite, nous détaillerons quelques méthodes, dans ce contexte ci, à savoir [Che05, Cai03, Tal07b]. Ces méthodes présentent comment les THDs peuvent être étendues pour traiter des requêtes complexes. Une bonne étude de ces méthodes figure dans le tutorial de Ranjan et al. [Ran08].

[Che05] propose une méthode de découverte de ressources de calcul capable de traiter des requêtes à intervalle et à multi-attribut. La méthode proposée, basée sur le système P2P Pastry, permet la découverte des métadonnées statiques (e.g. vitesse CPU, taille RAM) et des métadonnées dynamiques (e.g. charge CPU, RAM libre) décrivant les ressources de calcul. L'idée est de concaténer les métadonnées statiques avec les métadonnées dynamiques pour avoir un seul identificateur qui servira de clé pour les opérations des THDs. Afin de permettre une découverte à intervalle, la partie statique d'une ressource est mappé à un point fixe sur la THD tandis que la partie dynamique figure le long d'un arc (sur la THD), en allant de la partie statique, pour représenter toutes les métadonnées dynamiques possibles. Le stockage des métadonnées dynamiques dans les THD nécessite naturellement un protocole pour la maintenance de ces métadonnées. Dans ce contexte, [Che05] définit un protocole pour la mise à jour des métadonnées dynamiques en définissant un certain intervalle pour forcer la mise à jour. Par exemple, si la charge CPU varie d'au moins X% alors un message de mise à jour est envoyé. Il est possible que ce protocole génère des métadonnées périmées surtout s'il y a un churn dans le système. Ainsi, le protocole envoie des messages périodiques pour supprimer les métadonnées qui ne sont pas mis à jour.

[Cai03] propose un système P2P structuré, nommé MAAN, employé pour les services d'informations d'un système de grille. Ce système étend Chord [Sto01] pour traiter des requêtes à intervalle et à multi-attribut. En effet, Chord associe, pour chaque clé, un identifiant codé sur *m*

bits en utilisant une fonction de hachage SHA-1. Cette méthode permet une distribution uniforme des identifiants entre tous les nœuds du système. Par contre, le hachage SHA-1 détruit la localité des clés et ne permet pas de localiser des attributs à valeur numérique dans un intervalle. Ainsi, MAAN définit une fonction de hachage *H* qui permet de préserver la localité des identifiants générés. Pour plus de détails sur *H*, une définition et plusieurs théorèmes figurent dans [Cai03].

Requêtes à intervalle : Supposons qu'on cherche les valeurs *v* pour un attribut *a* compris dans un intervalle [*l* ;*u*]. L'application de Chord nous conduit à une clé *k*/ *H(l)=k*. Le nœud n_l qui a reçu la requête vérifie localement s'il possède des valeurs *v* (avec $l \leq v \leq u$) pour l'attribut *a* puis redirige le message de découverte avec les résultats éventuels *RES* à son successeur, qui à son tour vérifie s'il possède des valeurs *v* (avec $l \leq v \leq u$) pour l'attribut *a*. Puis, il ajoute ses résultats à *RES*. Le processus se répète jusqu' à arriver au nœud n_u. Si *X* est le nombre de nœuds entre n_l et n_u alors la complexité de la découverte, en terme de temps de réponse, est de *O(LogN+X).*

Requêtes à multi-attribut : Les requêtes à multi-attribut sont traitées, en étendant les requêtes à intervalle, suivant 2 stratégies :

- La première stratégie recherche les attributs d'une ressource d'une requête en décomposant d'une manière itérative cette requête en plusieurs sous-requêtes, chacune sur un attribut spécifique. Pour une requête à M-attributs (i.e. a_1, a_2, … a_n sont les attributs d'une même ressource), la complexité en terme de temps de réponse est de $O(logN + max(X_{i(1 \leq i \leq n)})$ *(resp.* $O(\Sigma(1 \leq i \leq n) (LogN + X_i))$) où X_i est le nombre de nœuds stockant des valeurs de a_i.

- La deuxième stratégie a pour but de réduire la complexité de la découverte de la première stratégie, en lançant uniquement la découverte sur l'attribut considéré dominant vu que la découverte des autres attributs est inutile dans ce cas. En effet, une requête de découverte à multi-attribut doit satisfaire toutes les sous-requêtes de chaque attribut. Soit X le résultat d'une requête multi-attribut, et X_i le résultat sur chaque attribut a_i, alors $X=\cap Xi$. L'idée est de découvrir uniquement les ressources X_K sur l'attribut dominant a_k. La complexité de la découverte, en terme de temps de réponse et du nombre de messages, est de $O(LogN+N*S_k)$ où S_k est le degré de sélectivité de l'attribut S_k. pour tous les attributs.

[Tal07b] propose une méthode, basée sur le système P2P Chord, pour le traitement des requêtes à intervalle et à multi-attribut. Les requêtes peuvent inclure des métadonnées dynamiques. Dans [Tal07b], chaque ressource est identifiée par *q* attribut. Une THD est définie pour chaque attribut. Si le système est constitué de *q* type de ressources, alors le nombre de THDs est de *p*q*. Ce type de Framework permet la gestion de plusieurs ressources. Les identificateurs d'une THD sont générés en concaténant le « type-ressource » avec « attribut-ressource ». Ce choix permettra de découvrir n'importe quel attribut pour une ressource donnée. Les attributs dynamiques ne sont pas insérés dans les THD car cela provoquerai des problèmes de performances surtout en présence des mises à jour fréquentes des ressources dynamiques. Les requêtes de découverte de ressources dynamiques sont exécutées suivant un certain mécanisme de diffusion dans les THDs. L'objectif est de contacter tous les nœuds du système via la THD. L'avantage de la diffusion dans les THDs est qu'elle permet de contacter tous les nœuds du système, tout en évitant les messages redondants. La diffusion est effectuée en parallèle en utilisant les successeurs dans les THDs. Dans Chord, chaque nœud possède *O(log(N))* successeurs (nommée fingers), ainsi la complexité de cette propagation en terme de temps de réponse est de *O(log(N))* sauts. La complexité en terme de nombre de messages est de *(N-1)* messages pour un système constitué de *N* nœuds.

Dans le but de réduire le nombre de messages générés, [Tal07b] propose une découverte de ressources dynamiques incrémentales. L'idée est de réduire le degré de parallélisme en redirigeant les messages vers *D* successeurs (i.e. fingers) au lieu de *O(log(N))* (avec $D \leq O(log(N))$). Lorsque $D= O(log(N))$ la stratégie incrémentale est identique à la stratégie parallèle. Si *D=1* (i.e. un nœud possède un seul successeur), on revient à une approche purement séquentielle.

5.2.3. Méthodes tenant compte du principe de la localité

Les THDs permettent une découverte distribuée de ressources en un nombre limité de sauts. Dans les méthodes présentées ci-dessus (et dans la plupart des méthodes de découverte de ressources classiques basées sur les THDs), la découverte est gérée par une seule THD globale (ou une THD par attribut) pour tout le système.

L'utilisation d'une seule grande[14] THD (ou une THD par attribut) pour la découverte de ressources dans tout le système ne prend pas en compte l'existence de plusieurs OVs. Pourtant, un système de grille est composé de plusieurs OVs [Fos02, Mas08, Tal05] qui sont souvent dédiés à un domaine d'application (e.g. biologie, pathologie). Ainsi, l'utilisation d'une seule THD peut avoir deux inconvénients [Sam09]. Le premier inconvénient est lié au fait que le nombre de messages générés lors de la connexion/déconnexion des nœuds dans une OV_{locale} peut être très dense et peut ainsi perturber la découverte de ressources dans les autres OV_i ($i \neq locale$). Le deuxième inconvénient est que l'utilisation d'une seule THD ne prend pas en compte le principe de la localité lors de la découverte de ressources. Le principe de la localité indique que les utilisateurs d'une grille informatique accèdent souvent à des ressources dans leur domaine d'application. Plus précisément, les utilisateurs tentent de travailler en groupe et d'accéder aux mêmes ensembles de ressource [Iam02]. La propriété de la localité est très importante à considérer si on traite les ressources de type sources de données, car en général, les utilisateurs d'une grille informatique accèdent souvent à des sources de données dans leur domaine d'application [Wat05].

Dans un but de considérer le principe de la localité, [Mis04] présente une méthode de découverte de ressources basée sur l'utilisation de plusieurs THDs. Chaque THD est dédiée à une organisation et les THDs sont interconnectées d'une manière hiérarchique. L'interconnexion entre les THDs est effectuée en utilisant un ensemble de nœuds prédéfinis dans une OV (e.g. des nœuds passerelles). Ainsi, un nœud passerelle peut appartenir à une ou deux organisations. Dans plusieurs scénarios, un système peut être constitué de deux niveaux de THDs : le premier niveau définit une THD par organisation et le deuxième niveau est une THD globale permettant l'interconnexion entre les organisations (voir Figure 5).

[14] Le terme « grande » signifie ici un très grand nombre de nœuds qui peut atteindre des dizaines de milliers de nœuds

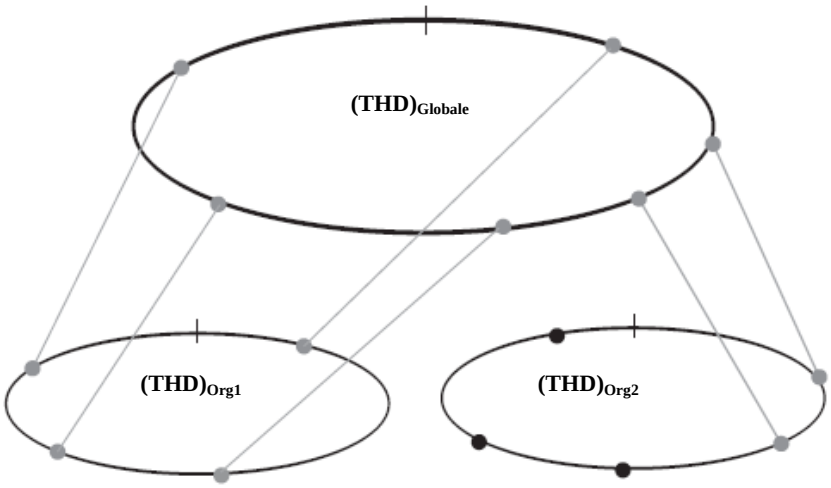

Figure 5 : Topologie P2P structurée hiérarchique à deux niveaux, inspirée de [Mis04]

Dans l'exemple de la Figure 5, tous les nœuds de l'organisation *Org1* appartiennent à la $(THD)_{Globale}$ tandis que dans l'organisation *Org2* uniquement les nœuds passerelles appartiennent à la $(THD)_{Globale}$. Lorsqu'une requête de découverte est soumise à une organisation, la découverte est d'abord effectuée dans l'organisation à laquelle appartient l'utilisateur. En cas d'échec, i.e. si la ressource à découvrir n'appartient pas à l'organisation locale, la requête de découverte est soumise au $(THD)_{Globale}$ via les nœuds appartenant à cette THD (i.e. n'importe quel nœud pour *Org1* et uniquement les nœuds passerelles pour *Org2*). Lorsqu'une ressource *R* d'une organisation *Orgi* veut publier ses métadonnées dans une organisation, une mise à jour classique est effectuée dans la $(THD)_{Orgi}$ [Sto01, Row01]. Pour que la ressource *R* soit visible au niveau de la THD globale, un message spécial est envoyé au THD global qui permet d'associer à *R* le nom de l'organisation à laquelle *R* appartient. Dans le cas où la ressource *R* est répliquée dans plusieurs organisations, toutes ces organisations seront spécifiées.

La création de deux niveaux de découvertes est suffisante dans un grand nombre de scénarios. Par contre, il existe des scénarios où la création de plusieurs niveaux de hiérarchie peut être utile afin d'affiner le principe de la localité. À titre d'exemple, l'université de Toulouse, constituée des trois grandes universités où chacune est composée de plusieurs organisations (e.g. laboratoires de recherche, bibliothèques universitaires), veut gérer les ressources en utilisant les THDs d'une manière hiérarchique dans une mini-grille. Dans un tel contexte, la création de deux niveaux d'hiérarchie n'est pas suffisante. Ainsi, une $(THD)_{UT}$ doit être créé pour l'Université de Toulouse, au premier niveau. Ensuite, les THDs suivantes doivent être créées $(THD)_{UT1}$, $(THD)_{UT2}$ et

$(THD)_{UT3}$ respectivement pour l'Université Toulouse 1, l'Université Toulouse 2 et l'Université Toulouse 3, au deuxième niveau. Enfin, la gestion de ressources des organisations (e.g. laboratoires) au sein d'une université est effectuée en définissant une THD pour chaque organisation (e.g. une $(THD)_{IRIT}$ sera définie pour l'IRIT de l'université Toulouse 3). Dans ce contexte, la méthode de [Mir04] étend la découverte de ressources pour un système de deux niveaux, à un système composé de *n* niveaux. Pour chaque ressource appartenant à une *Orgi*, une fonction de hachage est appliquée permettant de générer un identifiant avec le même préfixe pour toutes les ressources d'une même *Orgi*. Ainsi, lorsqu'une requête de découverte d'une ressource *R* est soumise à une *Orgi*, le système vérifie d'abord si le préfixe *h(R)* correspond bien à l'*Orgi* locale, sinon la requête est redirigée soit vers le niveau supérieur, soit vers le niveau inférieur jusqu'à arriver à l'*Orgi* concernée.

La complexité de la découverte dans une même *Orgi* est de *O(Log(Ni))* avec *Ni* le nombre de nœuds dans une *Orgi*. La propagation de la découverte inter-organisation est effectuée selon un mode unicast car les nœuds communs à deux ou plusieurs THDs sont connus à l'avance. La méthode est bien adaptée face à l'instabilité du système lors de la découverte intra-organisation (application classique des THDs). Par contre, la découverte inter-organisation peut poser problème si le(s) nœud(s) passerelle(s) d'une organisation est (sont) déconnecté(s). Dans ce cas, cette organisation ne peut plus communiquer avec d'autres organisations du système.

La maintenance du système peut devenir coûteuse car les nœuds passerelles doivent se connecter à plusieurs THDs ce qui nécessite un nombre important de messages de contrôle surtout pour la THD globale.

[Ali07] étend les travaux de [Mis04] en proposant une méthode pour la découverte de ressources de type calcul en traitant les requêtes de découvertes de ressources dynamiques, à intervalle et à multi-attribut.

5.2.4. Discussion

Les méthodes P2P structurées (les THDs) permettent un passage à l'échelle et elles sont bien adaptées face à l'instabilité des nœuds. Le second avantage de ces méthodes est que la découverte est fiable (indispensable pour les ressources de type source de données). La complexité, en terme de temps de réponse et du nombre de messages pour la découverte d'une ressource (e.g. un CPU,

un fichier XML) vaut, en général, *O(Log(N))* avec *N* le nombre de nœuds dans le système. D'où leur grand succès dans les systèmes de grille.

L'inconvénient majeur de ces méthodes est leur coût de maintenance, assez élevé. Par conséquence, le système est très dur à maintenir en présence de l'effet churn.

Le traitement de requêtes complexes (e.g. requêtes à intervalle et à multi-attribut) incluant uniquement de métadonnées statiques est devenu réalisable avec des systèmes tels que MAAN [Cai03], SWORD [Opp04], XenoSearch [Spe03]. Il est certain que la gestion des métadonnées dynamiques dans les THDs est assez délicate car cela nécessite une maintenance en continue des ressources (e.g. charge CPU, charge de la mémoire, bande passante). Si l'intervalle de la mise à jour des métadonnées dynamiques est trop court, cela peut provoquer une congestion dans le réseau à cause du nombre fréquent de messages envoyés. Dans le cas contraire, cela peut générer des métadonnées dynamiques périmées (e.g. la valeur d'une charge CPU peut changer de valeur rapidement). Le protocole proposé dans [Che05] est évalué par simulation et il semble acceptable pour la maintenance des THDs. Néanmoins, une vraie expérimentation dans ce contexte sera très appréciable afin de vérifier l'efficacité de la méthode proposée dans un système où les ressources peuvent changer d'état rapidement.

Dans un but de considérer l'autonomie des OVs et le principe de la localité, les travaux de [Ali07, Mis04, Gar03, Har03] ont été proposés. Nous sommes convaincus que la prise en compte du principe de la localité lors de la découverte de ressources est fondamentale surtout si les ressources considérées sont des sources de données. La découverte de ressources dans un système utilisant une seule THD (ou plusieurs THDs) sans considérer l'existence de plusieurs OVs dans le système n'est pas adaptée dans un système à très grande échelle et instable [Sam09]. Dans [Mis04, Ali07] Chaque THD est dédiée à une organisation et les THDs sont gérées d'une manière hiérarchique. L'inconvénient principal de ces deux méthodes est que la découverte inter-organisation n'est pas robuste en présence de la dynamicité des nœuds. Un deuxième inconvénient est lié à la maintenance de la THD globale qui nécessite un nombre très important de messages de contrôle dans un système à grande échelle.

5.3. Les méthodes super-pairs

5.3.1. Principe

Un super-pair est un nœud d'un réseau P2P qui joue le rôle d'un serveur pour un ensemble de nœuds clients et forme un nœud dans un réseau P2P non-structuré [Yan03]. Les réseaux super-pairs permettent une découverte hybride qui bénéficie de l'avantage de l'approche centralisée (e.g. maintenance facile car un seul point d'accès) et permettent une recherche distribuée. Dans les méthodes super-pairs [Fil04, Mas05, Pup05], le système est supposé être constitué de plusieurs OVs. Les OVs sont constituées d'un petit nombre de nœuds [Mas05].

5.3.2. Méthodes

Dans les méthodes super-pairs, chaque super-pair gère les métadonnées statiques et dynamiques décrivant les ressources (e.g. données, CPU, charge CPU) appartenant à son organisation virtuelle *OV* d'une manière centralisée ou hiérarchique [Mas05]. Les nœuds super-pairs sont organisés dans une couche pair-à-pair non structurée. Ils communiquent entre eux soit par diffusion, soit par des stratégies prédéfinies (e.g. empirique [Mas05], index de routage [Pup05]).

Une requête de découverte est d'abord soumise localement dans une *OV* (le principe de localité est alors appliqué) puis la découverte est propagée dans toutes les autres OVs (mode broadcast) ou vers un sous-ensemble d'OVs (mode anycast) [Mas05]. [Mas05] définit aussi un mécanisme TTL pour contrôler le nombre de sauts vers les autres OVs.

La complexité en terme de temps de réponse (resp. en terme du nombre de messages) est de $O(S+\Sigma C_{pi(1 \leq i \leq S)})$ (resp. $O(S^2)$) où S est le nombre des nœuds super-pairs dans le système et C_{pi} le coût du traitement local au niveau du super-pair *pi*. C_{pi} est le coût nécessaire pour la comparaison des métadonnées d'une requête de découverte avec les métadonnées décrivant les ressources gérées par un super-pair.

La maintenance des métadonnées de ressources d'une OV est gérée par le nœud super-pair, responsable d'une *OV*. Les métadonnées dynamiques au sein d'une OV sont maintenues en utilisant MDS-4 [Fos05].

Dans la Figure 6, nous présentons un exemple [Yan03] d'une topologie super-pair dans un système qui contient quatre OVs. Chaque cercle désigne une OV, le super-pair est présenté par un nœud noir tandis que les nœuds blancs sont des nœuds clients.

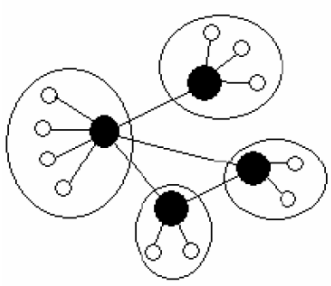

Figure 6 : Exemple d'une topologie super-pair [Yan03]

Les inconvénients de la méthode de [Mas05] sont : (i) la création d'un goulet d'étranglement si un grand nombre de messages est envoyé au nœud super-pair et (ii) la panne du nœud super-pair rend l'*OV* inutilisable et inaccessible par les autres *OVs* du système. Pour éviter (i) et (ii), [Pup05] propose la duplication d'un nœud super-pair en *K* super-pairs. Ceci peut augmenter le coût de la communication lorsqu'une ressource est ajoutée dans le système car cet ajout nécessite *K* messages pour informer tous les *K* super-pairs d'une même OV. D'autre part, la communication entre deux OVs coûte *O(K)* messages. Pire, si une OV veut communiquer avec *p* OVs, la complexité est de *O(p*K)* messages. La redondance d'un nœud super-pair augmente la complexité de la découverte en terme de nombre de messages mais pas en terme de temps réponse. Dans la Figure 7, nous présentons un exemple [Yan03] d'une topologie super-pair avec un degré de redondance de deux pour les super-pairs.

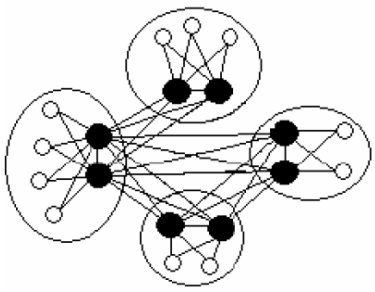

Figure 7 : Exemple d'une topologie super-pair avec un degré de redondance de deux [Yan03]

5.3.3. Discussion

Les méthodes super-pairs peuvent êtres efficace pour des grilles constituées d'OVs de petites tailles (e.g. une centaine de nœuds). Mais une OV doit pouvoir passer à l'échelle d'un grand nombre de nœuds. D'autre part, la panne du super-pair rend l'OV inutilisable et inaccessible par le système.

La complexité en terme de temps de réponse (resp. en terme du nombre de messages) est de $O(S+\Sigma C_{pi(1 \leq i \leq S)})$ (resp. $O(S^2)$) où S le nombre des nœuds super-pairs dans le système et C_{pi} est le coût du traitement local au niveau du super-pair *pi*.

La maintenance des métadonnées dans une OV est faite comme dans les approches hiérarchiques car le super-pair est implémenté avec MDS (i.e. MDS-4 dans [Mas05] et MDS-3 dans [Pup05]). Ces méthodes peuvent traiter des requêtes complexes. En effet, un super-pair peut implémenter les techniques nécessaires pour ce traitement en utilisant, par exemple, des services web. Ces méthodes peuvent traiter des requêtes de découverte des métadonnées dynamiques (MDS-4 est utilisé dans une OV).

6. Conclusion

Dans cette section, nous souhaitons effectuer une comparaison globale et conclure sur les catégories présentées ci-dessus. Le but de cette comparaison est de nous guider dans la conception d'une méthode efficace de découverte de sources de données.

Pour chaque catégorie/sous-catégorie, nous indiquons par un caractère «X» si les méthodes appartenant à une catégorie/sous-catégorie peuvent opérer en fonction des critères déjà présentés dans la section 2 (voir le Tableau 1). Nous indiquons la complexité dans un deuxième tableau (voir le Tableau 2) contenant le coût de la découverte associé à chaque catégorie/sous catégorie de méthode (les termes qui figurent dans le Tableau 2 figurent dans les sections consacrées pour chaque catégorie de méthode). Dans la suite, nous indiquons pour chaque catégorie, la meilleure sous-catégorie (si elle existe) en fonction des critères prédéfinis :

– Les méthodes hiérarchiques, basées sur une approche client/serveur, peuvent passer à une échelle nationale d'un système de grille. Cependant, le passage à une échelle mondiale reste un challenge. L'avantage de ces méthodes est que la découverte est fiable. C'est la raison pour laquelle elles ont été utilisées pour la découverte de sources de données dans plusieurs travaux de recherche [Hua04, Jay04] et notamment dans le projet OGSA-DAI [Ant05, Lyn09].

- Les méthodes basées sur des agents en mode coopération directe permettent un passage à l'échelle et peuvent opérer en présence de l'instabilité du système. Par contre, la découverte n'est pas fiable.

- Les méthodes P2P structurées (basées sur des THDs) sont meilleures par rapport aux méthodes non structurées et super-pairs. Elles passent à l'échelle, opèrent en présence de l'instabilité et permettent une découverte fiable. Le principal inconvénient est le coût de maintenance des THDs, assez élevé en présence de l'effet churn.

Critères / Catégories	Sous-catégories	Passage à l'échelle	Instabilité	Propagation de la découverte				Maintenance élevée (en présence du churn)	Découverte fiable	Requêtes complexes	
				Succession d'unicasr	Anycast	Multicast	Broadcast			Statique	Dynamique
Client/serveur	**Structure hiérarchique**			**X**				**X**	**X**	**X**	**X**
Agent	**Coopération hiérarchique**			**X**				**X**	**X**	**X**	**X**
	Coopération directe	**X**	**X**	**X**	**X**					**X**	**X**
P2P	**Non-structuré**		**X**				**X**		**X**	**X**	**X**
	Structuré (THD)	**X**	**X**	**X**				**X**	**X**	**X**	
	Super-pair					**X**			**X**	**X**	**X**

Tableau 1 Comparaison globale des méthodes de découverte de ressources

Catégories \ Critères	Sous-catégorie	Coût de la découverte	
		Temps de réponse	Nombre de messages
Client/Serveur	Structure hiérarchique	$O(H+\Sigma C_{i(1 \le i \le H)})$	$O(H)$
Agent	Coopération hiérarchique	$O(H+\Sigma C_{i(1 \le i \le H)})$	$O(H)$
	Coopération directe	$O(d+\Sigma C_{i(1 \le i \le d)})$	$O(d)$
P2P	Non-structuré	$O(d)$	$O(N^2)$
	Structuré (THD)	$O(Log(N))$	$O(Log(N))$
	Super-pair	$O(S+\Sigma C_{pi(1 \le i \le S)})$	$O(S^2)$

Tableau 2 Comparaison globale des coûts de découverte de ressources

D'après le Tableau 1, les méthodes basées sur des agents en mode coopération directe et les méthodes P2P structurées (basées sur des THDs) sont les méthodes les plus adaptées à un système de grille surtout par rapport au respect du passage à l'échelle et de l'instabilité du système.

Les méthodes basées sur des agents en mode coopération directe (e.g. [Kak06, Din05]) peuvent aider à la découverte de ressources. Elles peuvent aussi aider à la maintenance du système. En effet, le système d'information d'une grille n'est pas sensé stocker tous les « battements du cœur » [Yu06] de toutes les ressources du système. Ces méthodes permettent aussi de traiter des requêtes complexes incluant des métadonnées dynamiques. L'inconvénient majeur est que la découverte n'est pas fiable et que la complexité de celle-ci en terme de temps de réponse peut être très élevée sans aboutir à un résultat comme dans [Din05, Kak06]. Enfin, la structure interne de l'agent reste très complexe à implémenter surtout si plusieurs types d'agents existent dans le système. Un agent est sensé interagir avec d'autres agents ou d'autres composants dans le système (e.g. serveur LDAP [Yan07]) afin de prendre des décisions lors de la découverte de ressources (e.g. découverte au niveau du serveur local ou global [Yan07]). Ainsi, les agents doivent être en parfaite coordination et ils doivent stocker un minimum d'information pour éviter un surcoût lors de leur déplacement. Les méthodes basées sur des agents (en mode

coopération directe) ne sont pas bien adaptées pour la découverte de sources de données car la découverte n'est pas fiable.

Les THDs sont les plus adéquats pour un système de grille. Elles peuvent passer à l'échelle, opérer dans un système instable et garantir une découverte fiable en un nombre limité de sauts. Le défaut principal de ces méthodes est leur coût de maintenance surtout en présence de l'effet churn. Enfin, le traitement des requêtes complexes est devenu réalisable grâce à plusieurs systèmes (eg. MAAN [Cai03], SWORD [Opp04], XenoSearch [Spe03]). La manipulation des métadonnées dynamiques est très délicate car ces métadonnées nécessitent une mise à jour régulière, ce qui peut rapidement surcharger le réseau [Sam07, Tal07b]. Ainsi, il serait beaucoup plus efficace de stocker uniquement les métadonnées statiques dans les THDs. Les métadonnées dynamiques peuvent être récupérées grâce à un outil de monitoring (e.g. le Network Weather Service (NWS) [Wol99]) lors de la sélection de ressources [Sam07, Sam08] ou aussi lors de placement des tâches (e.g. opérateurs relationnels [Gou06, Smi02]).

Un point fondamental à considérer, si l'on adopte les THDs, est la prise en compte du principe de la localité (cf. section 5.2.3). Le principe de la localité est indispensable lors de la conception des applications dans un système à grande échelle. En résumé, ce principe précise que les utilisateurs accèdent souvent à des ressources qui résident dans leur domaine d'application surtout si les ressources considérées sont des sources de données. Ainsi, une méthode de découverte de sources de données doit « naturellement » s'exécuter dans l'organisation à laquelle appartient un utilisateur.

La décomposition de la THD globale en plusieurs THDs où chaque THD_i est associée à une OV_i du système est une solution attrayante [Sam09]. Par contre, comment : (i) permettre une interconnexion robuste entre toutes les OVs en présence de la dynamicité des nœuds et (ii) éviter un coût de la maintenance très élevé de l'interconnexion entre les THDs en présence de l'effet churn ? Une solution originale pour (i) et (ii) sera détaillée dans le chapitre suivant.

Chapitre III. Conception d'une méthode efficace de découverte de sources de données pour l'évaluation de requêtes en environnement de grille de données

1. Introduction

Dans le chapitre précédent, nous avons pu constater que le processus de découverte de ressources a été largement étudié dans les systèmes de grille durant ces dernières années. La plupart des méthodes de découverte de ressources sont centrées principalement sur la découverte de ressources de type calcul pour des grilles de calcul. La découverte de ressources de calcul consiste à découvrir un ensemble de ressources de calcul suivant un certain ensemble de critères (e.g. un nœud dont la vitesse CPU > 1 Ghz et dont les RAM > 2 Ghz) pour l'exécution d'une tâche. Dans ce contexte, un nœud dont la vitesse du processeur est de 2 Ghz peut être utilisé pour une tâche qui nécessite un processeur de 1 Ghz. Ceci ne peut pas être appliqué pour la découverte de sources de données qui possède une certaine particularité. La découverte de sources de données diffère principalement de la découverte de ressources de calcul par le fait qu'il faut qu'elle soit absolument fiable. En effet, les sources de données référencées dans une requête sont indispensables pour l'exécution de cette requête. Ainsi, une méthode de découverte de sources de données pour l'évaluation de requêtes sur des grilles de données doit être fiable, en tenant compte de l'instabilité et la grande échelle du système.

D'après le tableau de comparaison globale vu dans le chapitre précédent, nous avons pu constater que les méthodes basées sur l'utilisation des THDs sont les méthodes les plus adéquates pour la découverte de sources de données. Ces méthodes garantissent une découverte fiable, peuvent opérer en présence de l'instabilité du système et permettent un meilleur passage à l'échelle. Par contre, la maintenance des THDs nécessite un nombre important de messages lors

de la connexion ou de la déconnexion d'un nœud. En particulier, le nombre de messages générés peut être très dense dans le cas d'un effet churn.

La méthode proposée dans ce chapitre est basée sur l'utilisation des THDs. L'idée est de décomposer la THD globale [Dov03, Gal03, Kin07], souvent utilisée pour la découverte de ressources, en plusieurs THD_i où chaque THD_i est dédiée à une Organisation Virtuelle *OVi* dans le système. Deux niveaux de découverte sont définis : intra-OV et inter-OV. Lorsqu'un utilisateur soumet une requête de découverte de source de données, la découverte est d'abord lancée dans l'OV locale (i.e. à laquelle l'utilisateur appartient) car nous favorisons le principe de la localité de données. La découverte intra-OV est une découverte classique, basée sur les THDs, permettant une découverte fiable en un nombre limité de sauts. Si la découverte locale échoue, alors la découverte inter-OV est exécutée dans toutes les autres OV_i ($i \neq locale$) du système (car les ressources considérées sont des sources de données indispensables pour l'exécution d'une requête). La découverte inter-OV est aussi fiable car chaque OV_i distante dispose aussi de sa propre THD_i. L'interconnexion entre les différentes OVs est effectuée grâce à un système d'adressage simple mais original qui permet un accès permanent de n'importe quel nœud d'une OV_{locale} à toute autre OV_i (avec $i \neq locale$) avec un faible coût de maintenance en présence de l'effet churn. La maintenance du système d'adressage est effectuée lors de la découverte de sources de données.

Le reste du chapitre est organisé comme suit. Dans la section 2, nous proposons une architecture d'évaluateur de requêtes pour l'évaluation de requêtes SQL en environnement de grille de données. Le but de cette architecture est surtout de situer la phase de découverte de sources de données, la phase de découverte de ressources de calcul et enfin le monitoring de ressources de calcul. Nous présentons les différentes interactions entre les différents modules de cette architecture et les deux services : le Service de Découverte de Ressources (SDR) [Sam07, Sam09] et le service de monitoring de ressources, le Network Distant Service (NDS) [Gos07a]. Dans la section 3, nous détaillons le problème abordé dans ce chapitre puis nous proposons une méthode de découverte de sources de données pour l'évaluation de requêtes en environnement de grille de données [Sam09]. Nous clarifions, par plusieurs illustrations, le comportement de la méthode proposée, en l'absence puis en la présence de la dynamicité des nœuds. Enfin, nous concluons ce chapitre dans la section 4.

2. Une architecture d'évaluateur de requêtes SQL en environnement de grille de données

Dans cette section, nous présentons une architecture d'évaluateur de requêtes SQL en environnement de grille de données. Nous décrivons les différentes phases d'évaluation d'une requête SQL dans l'architecture proposée en précisant le rôle de chaque module. L'hétérogénéité de données (e.g. fichiers, programmes d'application) est abstraite au moyen d'un système d'intégration virtuelle de données basé sur une architecture médiateur-adaptateur [Wie92]. Nous ne traitons pas l'hétérogénéité sémantique et structurelle [Kin09].

Dans la littérature, plusieurs systèmes de médiation de données ont été proposés [Man01, Haa99, Gar97]. Ces systèmes proposent des solutions aux problèmes liés à l'autonomie et à l'hétérogénéité des sources de données en fournissant une vue virtuelle des données. Ainsi, les utilisateurs interrogent le système de médiation, sans avoir besoin de connaître les méthodes d'accès aux sources de données. Les utilisateurs formulent leurs requêtes en terme de schéma global de médiateur. Les principaux composants d'un système de médiation sont : (i) les médiateurs, (ii) les adaptateurs et (iii) les sources de données. Un adaptateur (i) agit comme un intermédiaire entre une source de données et des médiateurs. Il transforme les requêtes soumises par les médiateurs en requêtes exécutables par la source. L'adaptateur assure la connexion à la source. Il est responsable de l'envoi des requêtes à la source, de la récupération des résultats des requêtes et de la transformation des résultats en un modèle de données des médiateurs. Il cache ainsi l'hétérogénéité des sources de données. Le rôle principal d'un médiateur (ii) est d'abstraire, d'intégrer, d'exploiter et de simplifier l'accès aux données provenant de plusieurs sources. Il fournit un schéma global de données et des interfaces d'interrogation permettant aux utilisateurs d'accéder aux informations intégrées. Ainsi, les utilisateurs reformulent leurs requêtes en terme de schéma global de données. Une source de données (iii) peut être définie comme un triplet : <données, méthodes d'accès, moteur d'exécution>. En plus de la recherche de données, les méthodes d'accès permettent d'effectuer des statistiques sur les données issues de sources et d'accéder aux métadonnées décrivant ces données. Une source de données (iii) peut être : un ensemble de fichiers XML, une base de données relationnelle gérée par un SGBD (e.g. Oracle, DB2), un système d'information d'une organisation …etc. Dans la suite du document, nous utilisons le terme nœud au lieu de médiateur, pour garder la cohérence dans l'ensemble du

document. Nous nous basons sur un modèle relationnel vu que ce dernier a été largement utilisé dans le monde des bases de données. Ainsi, les sources de données traitées sont des relations.

Dans la suite, nous présentons les différentes phases de l'évaluation de requêtes SQL dans l'architecture proposée. Nous supposons qu'un utilisateur soumet sa requête via une interface et en passant par des mécanismes d'authentification (prédéfinies par l'administrateur de la grille). Enfin, une instance de cette architecture tourne sur chaque nœud du système. Les différentes phases de l'évaluation d'une requête SQL *Req* sont les suivantes :

(a) **Découverte de relations :** La première phase consiste à découvrir les métadonnées décrivant l'ensemble des relations $E=\{R_i ; \ldots ; R_m\}$ référencées dans *Req*. Les paramètres d'entrée de cette phase correspondent aux noms des relations qui figurent dans *E*. Les paramètres de sortie contiennent un ensemble de métadonnées $M = \{MetadataR_i ; \ldots ; MetaDataR_m\}$ où $MetaDataR_k$ contient les métadonnées décrivant de R_k. Plus précisément, pour une relation R_k, les métadonnées envoyées concernent: (i) le profil de R_k et (ii) le placement de R_k. Dans le profil de R_k, on trouve le schéma relationnel de R_k (i.e. nom des attributs de R_k et leur type) et des valeurs statistiques sur R_k (e.g. la valeur minimale d'un attribut et le nombre des valeurs distincts). Ces valeurs statistiques sont utilisées lors de la phase d'optimisation. Quant au placement de R_k, il contient les informations liées à la localisation, la fragmentation et la duplication de R_k.

(b) **Agrément (relations):** Il est important d'effectuer une phase d'agrément dans un système instable. L'agrément signifie qu'on vérifie si au moins (dans le cas où une relation est répliquée sur plusieurs nœuds) un nœud stockant une relation (ou le fragment d'une relation) est bien connecté au système et qu'il est possible d'atteindre ce nœud. En effet, il serait mieux d'arrêter l'évaluation de la requête à cette phase et de renvoyer un message d'échec à l'utilisateur si un nœud stockant une relation ou un fragment n'est disponible.

(c) **Décomposition de requêtes :** l'entrée de cette phase est une requête distribuée SQL. Comme dans les systèmes de bases de données distribuées [Ozs99], la décomposition des requêtes subit quatre étapes successives : (i) la normalisation, (ii) l'analyse, (iii) l'élimination des redondances et (iv) la réécriture. Les étapes (i), (iii) et (iv) s'appuient sur

un ensemble de règles de transformation d'une requête afin d'avoir de meilleures performances. Par contre, l'étape (ii) s'appuie sur une partie de l'ensemble *M*. Il s'agit de la partie concernant le profil des relations référencées dans *Req*, plus précisément les attributs de ces relations. En effet, l'étape d'analyse est responsable d'effectuer l'analyse syntaxique et sémantique. L'analyse syntaxique permet de vérifier si le nom d'un attribut (d'une relation) référencé dans *Req* figure bien dans le schéma relationnel (e.g. cette étape vérifie si l'attribut *nomD* figure bien dans le schéma relationnel d'une relation *Docteur*). Elle permet aussi de vérifier si une comparaison est possible (e.g. un attribut de type entier doit être comparé exclusivement avec un entier). Quant à l'analyse sémantique, elle permet de vérifier, par exemple, si la relation *Docteur* désigne un médecin ou un docteur en science. Nous ne traitons pas le problème de l'hétérogénéité sémantique [Kin09] dans cette thèse. La sortie de cette phase est une requête algébrique « bonne » dans le sens où les requêtes incorrectes sont rejetées, sans redondance, exprimée sur des relations globales.

(d) **Localisation des données :** L'entrée de cette phase est une requête algébrique exprimée sur des relations distribuées. La localisation des données est effectuée en fonction d'une partie des métadonnées qui figure dans l'ensemble *M*. Plus précisément, il s'agit des métadonnées concernant le placement des relations. Comme dans les systèmes de bases de données distribuées [Ozs99], la localisation des données consiste à la transformation d'une requête répartie (exprimée sur des relations globales) en une requête répartie équivalente exprimée sur des fragments. Plus précisément, la localisation d'une requête répartie se déroule en deux étapes [Gar90]: (i) la génération d'une requête canonique équivalente et (ii) la simplification. La requête canonique (i) est générée à partir de la requête répartie exprimée sur des relations globales en remplaçant chaque relation par sa requête de reconstruction correspondante. La reconstruction d'une relation à partir de ses fragments est effectuée par une opération relationnelle (e.g. une jointure). La simplification (ii) consiste à supprimer les opérations inutiles qui peuvent, par exemple, produire un résultat vide.

(e) **Découverte de ressources de calcul :** Cette phase consiste à découvrir uniquement les métadonnées statiques décrivant les ressources de calcul [Sam07]. Il s'agit de la découverte d'un sous-ensemble des nœuds disponibles pour l'exécution d'une requête. Il est plus efficace de stocker des métadonnées statiques dans les THDs vu que le stockage des métadonnées dynamiques est très délicat [Sam07, Tru07] et peut provoquer des problèmes de performance comme nous l'avions mentionné dans le chapitre précédent. D'autre part, les métadonnées dynamiques ne sont pas utiles lors de cette phase. Ces métadonnées sont utiles pendant la phase d'optimisation. En effet, l'optimiseur peut décider de récupérer une partie de ces métadonnées (e.g. pour effectuer une opération de lecture, l'optimiseur demandera le nœud ayant des E/S moins chargés si plusieurs nœuds stockent une réplication d'une relation). Dans ce contexte, une stratégie simple consiste à découvrir les ressources de calcul (e.g. CPU ou mémoire d'un nœud) « les plus proches »[15] en utilisant les THDs et, en fonction de la localisation des nœuds stockant les relations découvertes [Sam07].

(f) **Agrément (ressources de calcul):** Parallèlement à la phase d'agrément de relations, il est important d'effectuer une phase d'agrément pour les ressources de calcul afin de vérifier si ces derniers sont bien connectés au système. Cette phase a pour but d'éliminer les ressources de calcul découvertes non disponibles suite à un évènement tel qu'une panne réseau. Les ressources de calcul que l'on a pu atteindre, sont « possiblement » allouables pour l'exécution d'une opération relationnelle (e.g. une jointure).

(g) **Optimisation :** L'objectif de l'optimisation de requêtes est de déterminer une stratégie d'exécution de la requête qui minimise une fonction coût. La phase d'optimisation inclut : (i) la sélection des meilleures réplications de relations, (ii) la sélection des meilleurs nœuds parmi l'ensemble des nœuds découverts, (iii) la définition de l'ordre des opérations relationnelles et le meilleur algorithme d'accès pour chacun d'eux, et (iv) le placement des opérateurs relationnels (appelé aussi allocation de ressources) pour l'exécution des opérations. L'optimiseur s'appuie sur un modèle de coûts composé essentiellement (i)

[15] Les ressources de calcul plus proches sont les ressources choisis suivant une métrique prédéfinie (e.g. la bande passante, le nombre de sauts)

d'un ensemble de métriques (e.g. temps de réponse d'une jointure) et (ii) d'une bibliothèque. Les métriques (i) sont calculées en fonction des valeurs des paramètres hôtes (i.e. valeur de la charge CPU, valeur de la charge des E/S et valeur de la charge de la mémoire d'un nœud) et réseaux (i.e. valeur de la latence et de la bande passante entre deux nœuds) stockées dans la bibliothèque. La bibliothèque (ii) contient des statistiques sur le système et sur les relations, et des formules de coûts pour estimer le coût d'exécution des requêtes. Plusieurs approches sont proposées pour l'estimation du coût d'exécution d'une requête (e.g. l'approche par calibration[16] [Du95, Gar96], l'approche par échantillonnage [Zhu98] ou aussi dans l'approche historique [Ada96]). Le principal problème, dans la plupart de ces approches, est qu'elles ne prennent pas en considération la variation des valeurs des paramètres hôtes et réseaux durant le calcul du coût d'exécution d'une opération relationnelle (e.g. une jointure). Ceci peut posséder des limites dans un système où les ressources sont partagées à une échelle mondiale et où chaque utilisateur (e.g. une application) peut se connecter à un nœud distant pour effectuer des tâches (e.g. soumission d'une requête).

Pour prendre en considération la variation de valeurs de paramètres hôtes et réseaux, nous proposons l'utilisation d'un service de monitoring capable de calculer la meilleure valeur estimée d'un paramètre hôte ou réseau. L'utilisation des informations de monitoring permet d'améliorer l'exactitude des métriques fournies par le modèle de coûts.

(h) **Exécution :** Dans la dernière phase, nous reprenons les travaux de [Arc04, Hus05a, Mor03] qui définit un modèle d'exécution à base d'agents mobiles pour l'optimisation dynamique de requêtes réparties à grande échelle. L'idée est d'exécuter chaque opérateur relationnel au moyen d'un agent mobile ce qui permet de décentraliser les décisions prises par l'optimiseur et de s'adapter dynamiquement aux erreurs d'estimation concernant le profil de relation. Dans [Arc04, Hus05b], un agent mobile utilise des paramètres hôtes et réseaux calibrés lors de la prise de ses décisions. Un paramètre calibré suppose d'avoir la même valeur entre deux instants. Ceci peut conduire à de mauvaises performances vu que ces paramètres peuvent changer de valeur d'une manière soudaine entre deux instants dans un système de grille. Dans ce contexte, nous proposons une méthode qui permet de

[16] Nous revenons d'une manière plus détaillé sur la définition des approches (e.g. par calibration) utilisés pour la définition des modèles de coûts dans le chapitre suivant.

prendre en considération la variation des valeurs des paramètres hôtes (e.g. valeur de la charge CPU d'un nœud) et réseaux (e.g. valeur de la charge de la bande passante entre deux nœuds) durant le calcul du coût d'exécution d'une opération relationnelle. L'idée est de permettre aux agents mobiles de capturer la meilleure valeur estimée d'un paramètre hôte et réseau en utilisant les informations de monitoring pendant l'exécution. La méthode proposée permet une meilleure prise de décision aux agents lors de leur migration et améliore ainsi la qualité des décisions de l'optimiseur [Sam08].

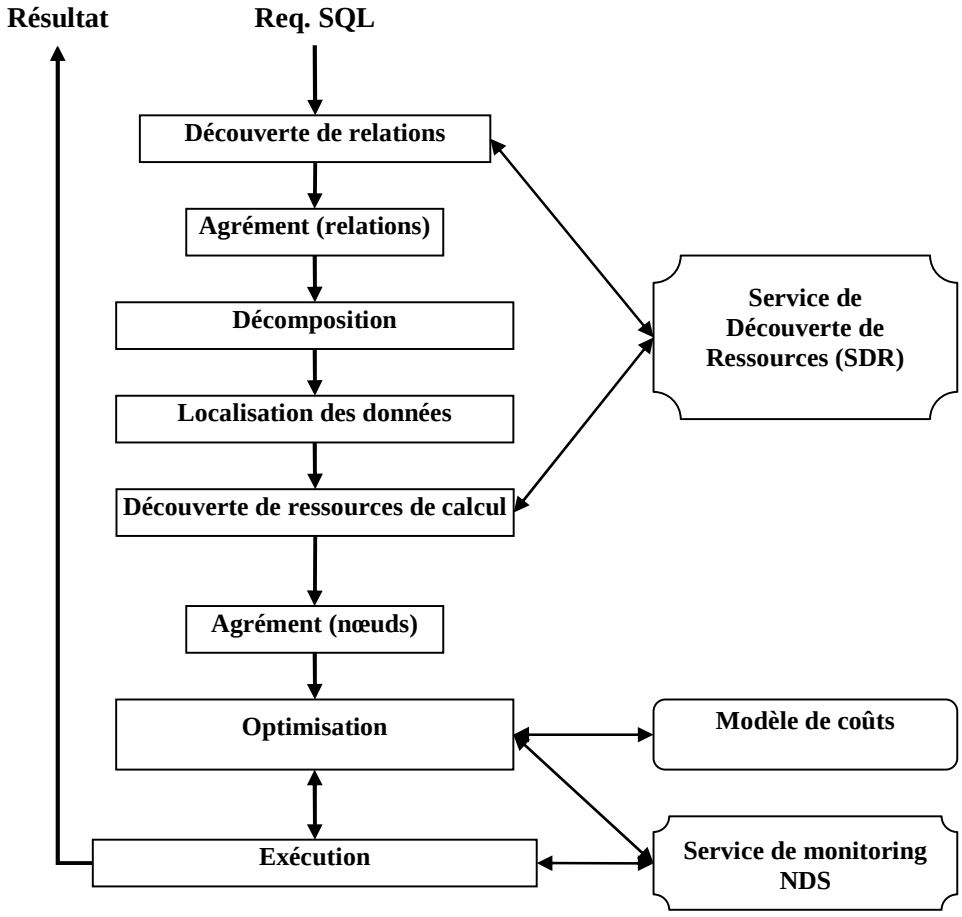

Figure 8 : Architecture d'évaluateur de requêtes SQL en environnement de grille de données

3. Découverte de relations pour l'évaluation de requêtes en environnement de grille de données

3.1. Introduction

Dans la suite, nous présentons, d'une manière plus précise, la problématique abordée lors de la découverte de relations pour l'évaluation de requêtes distribuées en environnement de grille de données. Ensuite, nous proposons une méthode [Sam09] dans ce contexte.

Nous supposons qu'une grille de données est composée d'un ensemble d'OVs [Iam01, Iam04, Mas05, Tal05]. Chaque OV est dédiée à un domaine d'application. Par exemple, une OV est dédiée au domaine de la physique et une autre OV est dédiée au domaine de la biologie. Nous rappelons qu'une OV est constituée d'un grand nombre de nœuds (des milliers de nœuds) et que chaque nœud peut joindre ou quitter le système à tout instant. Une OV peut aussi se déconnecter à tout instant du système.

3.2. Position du problème

L'utilisation d'une seule THD ne prend pas en compte le principe de la localité lors de la découverte de ressources. Ce principe indique que les utilisateurs d'une grille Informatique accèdent souvent aux mêmes ensembles de ressources [Iam02], dans leur OV associée à un domaine d'application. En particulier, les utilisateurs (e.g. biologistes, physiciens, pathologistes) d'une grille de données demandent souvent l'accès aux sources de données (ici, ce sont des relations) [Wat05] dans leur propre OV. Pour favoriser la localité de données, deux niveaux de découverte peuvent être dégagés : intra-OV et inter-OV.

Lorsque le système reçoit une requête de découverte d'une relation *R*, la découverte intra-OV de *R* est exécutée dans l'OV_{locale} à laquelle appartient l'utilisateur qui a soumis la requête (favorisation du principe de la localité). La découverte intra-OV est une découverte classique basée sur les THDs qui permet de garantir une découverte fiable en un nombre limité de sauts. Si la découverte dans la THD_{locale} (associée à l'OV_{locale}) échoue, le problème est de savoir comment lancer la découverte de *R* dans toutes les OV_i distantes (avec $OV_i \neq OV_{locale}$). Il faut surtout éviter d'utiliser le même nœud (i.e. central) comme dans les méthodes super-pairs [Pup05, Mas05]. En d'autres termes, comment permettre un accès permanent d'un nœud *N* d'une OV_{locale} à toute autre OV_i ($i \neq locale$) en présence de la dynamicité des nœuds ? Nous proposons d'associer à chaque nœud *N* d'une OV_{locale} un point de sortie N_i vers toute autre OV_i avec ($i \neq locale$). Les points de

sortie (N_i)s ne sont pas tous identiques dans une même OV_{locale}. Ainsi, on peut toujours trouver, à partir d'une OV_{locale}, un point de sortie à tout autre OV_i ($i \neq locale$). L'interconnexion entre toutes les OVs est effectuée grâce à un système d'adressage qui est mis à jour lors de la découverte de relations, ayant un faible coût de maintenance en présence de l'effet churn. Nous allons décrire d'une manière approfondie comment s'effectue la découverte de relations (intra-OV et inter-OV).

3.3. Découverte de relations

3.3.1. Découverte intra-OV

La découverte intra-OV est une découverte classique, basée sur les THDs. Une THD permet le stockage et l'accès aux pairs de type <clé, valeur>. Elles associent pour chaque clé, une valeur. L'accès à une clé est fait en *O(Log(M))* étapes avec *M* le nombre des nœuds dans le système. Pour cela, les THDs proposent des primitives telles que : *lookup(clé)* (retournant la valeur associée à la clé dans la THD) et *Store(clé, valeur)* (qui permet d'associer une *valeur* pour chaque *clé*). D'une manière générale, la clé recherchée dans les systèmes P2P classiques est le nom d'un fichier, la valeur renvoyée est l'adresse IP du nœud (i.e. la machine) qui stocke ce fichier. La valeur renvoyée peut inclure, en plus de l'adresse IP, d'autres métadonnées (e.g. la taille d'un fichier) en utilisant un système tel que Past (une extension de Pastry) [Row01].

Dans notre système, la découverte de métadonnées *metaDataR* décrivant une relation *R* est exécutée avec la primitive *lookup(R,*OV_{locale}*)* avec OV_{locale} correspondant à l'OV où la découverte est lancée. Si la découverte intra-OV de *R* échoue alors la découverte doit être effectuée dans toutes les autres OVs appartenant au système. Cette action est nommée la découverte inter-OV. La valeur renvoyée contient les métadonnées *metaDataR* concernant : (i) le profil de *R* et (ii) le placement de *R*. Les métadonnées (i) contiennent : le nom de chaque attribut de *R* et son type. Les métadonnées (ii) incluent : la localisation des fragments de *R* (i.e. l'adresse de chaque nœud qui stocke un fragment de *R*) et la règle de reconstruction de *R*. Les métadonnées sont envoyées sous ce format :

MetaDataR = {Att = (att1 type_att1[, att2 type_att2, …]) ;
SP=(R1(N_{i1}*[,*N_{i2}*, …])[, R2(* N_{j1}*[,*N_{j2}*, …)], …]) ;*
RR}

Le champ *Att* précise le nom de chaque attribut de *R* et son type. Le champ *SP* indique le schéma de placement de *R*, *R* pouvant être constitué de *R1*, *R2* et d'autres fragments. *R1* réside sur N_{i1} et peut être éventuellement répliqué en N_{i2} et d'autres nœuds (le signe « *[..]* » indique que c'est un champ éventuel). Il est en de même pour *R2*. Enfin, le champ *RR* indique la règle de reconstruction de *R*. Un parseur composé des trois méthodes *getAttributs(metaDataR)*, *getSP(metaDataR)* et *getRR(metaDataR)* est défini afin de récupérer, respectivement, les champs *Att*, *SP* et *RR* d'une relation.

Par exemple, soit une relation *Medecin* constitué de deux fragments, le premier *MedecinGeneraliste* est répliqué sur *N10* et *N50*. Le deuxième *MedecinSpecialiste* est répliqué sur *N12*, *N15 et N40*. La règle de reconstruction de médecin est une jointure sur l'attribut *noMedecin*. Les métadonnées *metaMedecin* sont stockées au niveau du nœud responsable de la relation *Medecin* sous ce format :

MetaMedecin = { Att = (noM entier, nomM char(10), adresseM char(100)) ;
SP = (MedecinGeneraliste(N10,N50), MedecinSpecialiste(N12,N15,N40));
RR}

Dans le cas où la découverte intra-OV échoue alors il faut lancer la découverte dans toutes les autres OVs (il s'agit alors d'une découverte inter-OV).

3.3.2. Découverte inter-OV

Lors d'une découverte inter-OV, le problème est de savoir comment interconnecter les OVs entre elles afin de pouvoir propager la découverte à toutes les autres OVs (≠ de l'OV locale) du système. Ceci est fait grâce à un système d'adressage permettant l'interconnexion entre toutes les OVs du système. L'idée est d'associer à chaque nœud *N* d'une OV_{locale} l'adresse d'un nœud N_i d'une OV_i appartenant au système. Le nœud N_i sera nommé le point de sortie vers l'OV_i. Les points de sortie vers une même OV_i ne sont pas tous identiques afin d'éviter les inconvénients d'un nœud central, i.e. la panne du nœud central et le goulet d'étranglement que peut former un nœud central si un grand nombre de messages est envoyé vers ce nœud. Ainsi, chaque nœud *N* d'une OV_{locale} ne dispose pas du même point de sortie (que ses voisins dans l'OV_{locale}) vers une OV_i $(i \neq locale)$. Un nœud *N* d'une OV_{locale} connaît un seul point de sortie vers chaque OV_i. Ainsi,

le nombre de points de sortie pour un nœud quelconque vaut le nombre total d'OVs dans le système moins un.

3.3.3. Algorithme de découverte de relations (intra-OV et inter-OV)

Durant la découverte intra-OV, le dernier nœud contacté *currentNode* dans la THD locale constate l'échec de cette découverte. Ainsi, *currentNode* initialise le processus de la découverte inter-OV en envoyant des messages aux autres OVs (≠ l'OV_{locale}) grâce à ses points de sortie *ps*. Si *currentNode* contacte un point de sortie ps_i vers une OV_i qui ne répond pas au bout d'un certain laps de temps (e.g. un Round Trip per Time RTT par exemple) alors *currentNode* contacte un voisin[17] à lui pour lui demander un autre point de sortie ps_j à l'OV désirée. Ceci est répété récursivement en demandant à mon voisin puis au voisin de mon voisin jusqu'à trouver un point de sortie connecté. Une fois trouvé, *currentNode* met à jour son point de sortie ps_i en demandant le voisin du point de sortie connectée (i.e. voisin(ps_j)) dans l'OV_i distante. Cette stratégie garantit que les points de sortie vers l'OV_i distante ne sont pas tous identiques.

Il est important de vérifier si on a parcouru tous les nœuds de l'OV_{locale} pour éviter des recherches en boucle pour un point de sortie. Ceci est fait en vérifiant, lors de chaque itération, si le voisin de mon voisin est le nœud courant *currentNode*. Si c'est le cas, l'OV_i distante est considérée comme déconnectée et l'information sur la déconnexion d'OV_i sera uniquement propagée dans OV_{loc}. Nous justifions ce choix dans la section nommé « Maintenance du système ».

La complexité de l'algorithme proposé en terme de temps de réponse est de $O(Log(M_i))$ + $coût\_accès(ps_i)$ avec *Mi* le nombre moyen de nœuds dans une OV_i et $coût\_accès(ps_i)$ est le coût moyen nécessaire pour trouver un ps_i connecté vers une OV_i. Ce coût dépend du nombre de voisins contactés dans une même OV et du coût pour envoyer un message vers une OV_i distante. La complexité de l'algorithme proposé en terme de nombre de messages est de (K-1) * $O(Log(M_i))$ + $\sum coût\_accès(ps_i)_{(1 \le i \le k-1)}$ (i est l'indice de l'OV_i) avec K le nombre d'OVs dans le système). L'algorithme de la découverte de relations est décrit dans la Figure 9.

[17] Nous utilisons le terme voisin pour désigner le prochain voisin (le plus proche) dans la THD

```
// R : une relation
// ps_i : point de sortie à OV_i
// OV_locale : OV locale
// OV_i : OV distante i

metaDataR = currentnode.lookup(R,OV_locale);
Si < metaDataR figure dans OV_locale >  alors Envoyer metadataR;
Sinon
{
      Pour chaque ps_i ∈ currentNode
      {
            Si < ps_i n'est pas connecté > alors
            {
                  //Cette recherche est effectuée récursivement en
                  // demandant à mon voisin puis au voisin de mon
                  // voisin et ainsi de suite
                 Demander un nouveau ps_j ∈ OV_i à mon voisin V
                 //vérifie si on a parcouru tous les nœuds d'OV_locale
                 et vérifier si V est le dernier nœud courant
                 si c'est le cas
                 {
                       //propager l'information sur la déconnexion de OV_i
                       // est uniquement dans l'OV_locale
                       propager(«OV_i est déconnectée» ; OV_locale)
                 } ;

                 // mise à jour du ps_i
                  ps_i ← voisin (ps_j) ;
            }
            metadataR ← ps.lookup(R,OV_i);
           Si < metaDataR figure dans OV_i >  alors
                  Envoyer metadataR;
           Sinon
                 Envoyer « la découverte de R a échoué » ;

      }
}
```

Figure 9 : Algorithme de découverte de relations

3.3.4. Stratégie de mise à jour paresseuse pour le système d'adressage

Dans l'algorithme de la Figure 9, la mise à jour du système d'adressage concerne uniquement le points de sortie ps_i du nœud *currentNode* qui initie le processus de la découverte inter-OV. Tous les autres points de sortie (récupérés via le voisin, puis le voisin de voisin de *currentNode*) vers une OV_i qui sont déconnectés ne sont pas mis à jour pour les raisons suivantes : si un grand nombre de nœuds *N_Dec* se déconnecte d'une OV_i, il y a une chance que l'ensemble de points de sortie *Ens_PS* d'une OV_{locale} utilise un sous-ensemble de *N_Dec* comme des points de sortie vers

OV_i. Dans un tel contexte, la mise à jour de *Ens_PS* peut être très coûteuse et surtout inutile si les nœuds de l'OV_{locale} disposant d'un point de sortie appartenant à *Ens_PS* n'a jamais eu besoin d'accéder à OV_i.

L'objectif est alors d'éviter un surcoût de maintenance du système d'adressage en présence d'un effet churn. Ainsi, nous employons une stratégie de mise à jour paresseuse pour le système d'adressage. Le système d'adressage est alors mis à jour lors de la découverte inter-OV et possède un coût de maintenance minimal en présence de l'effet churn.

3.3.5. Illustration : Découverte inter-OV

Pour illustrer le mécanisme de la découverte inter-*OV*, soit une grille composée de trois OVs dans le domaine : Biologique OV_{Bio}, Pathologique OV_{Pat} et Physique OV_{Phy} (Figure 10). Chaque THD_i est responsable de la gestion des métadonnées des relations stockées dans une OV_i. Pour la clarté de la figure, nous montrons uniquement les points de sortie de l'OV_{Bio} aux autres OVs.

Soit un utilisateur *U_Bio* de l'OV_{Bio}. *U_Bio* a besoin d'accéder à une relation *Alzheimer* (qui appartient normalement à l'OV_{Pat}). Le processus de la découverte intra-*OV* est lancé, par exemple, à partir du nœud *N23*. La découverte de cette relation échoue dans l'OV_{Bio} (à laquelle appartient *U_Bio*). Le nœud *N15* s'aperçoit que *Alzheimer* ne figure pas dans la $(THD)_{Bio}$. Dans ce type de scénario, le processus de la découverte inter-*OV* est initié par *N15*. Ainsi, *N15* envoie un message aux autres *OVs* grâce aux points de sortie : *N7* pour l'OV_{Pat} et *N5* pour l'OV_{Phy}. La découverte de *Alzheimer* est lancée simultanément dans les OVs distantes : l'OV_{Pat} et l'OV_{Phy} (il s'agit d'une découverte classique basée sur des THDs dans OV_{Pat} et OV_{Phy}). L'OV_{Pat} communiquera *metaDataAlzheimer* via le nœud responsable de la relation *Alzheimer*. La découverte dans l'OV_{Phy} échouera.

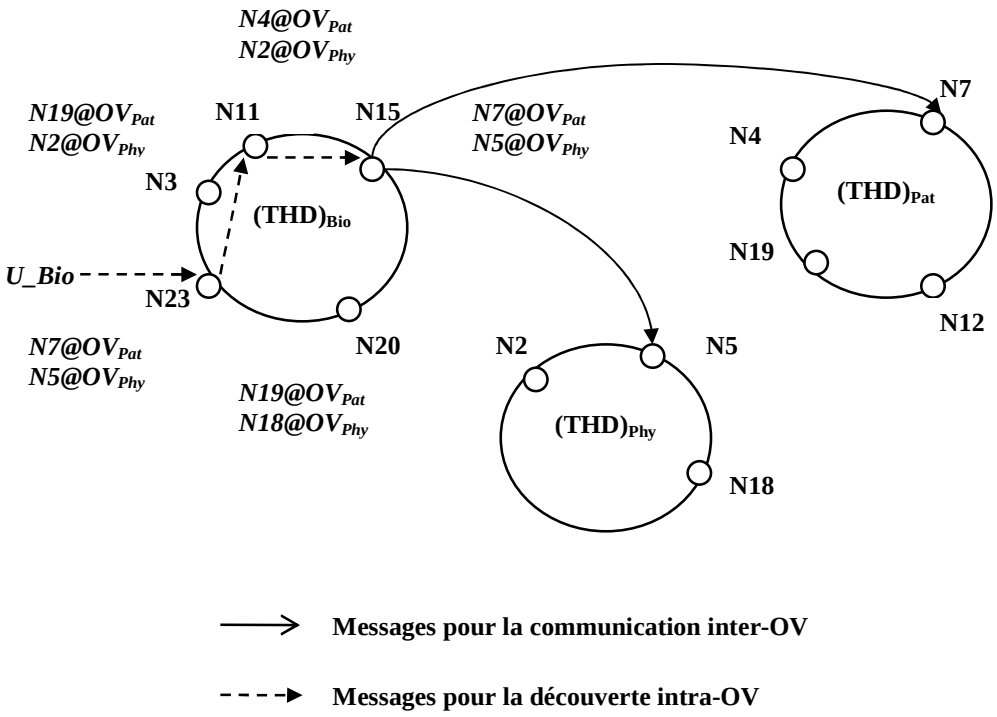

Figure 10 : Découverte inter-OV en l'absence de la dynamicité des nœuds

Dans l'exemple précédent, les points de sortie de *N15* : *N7* (vers l'OV_{Pat}) et *N5* (vers l'OV_{Phy}) sont supposés être connectés. Supposons maintenant que le point de sortie *N7* est déconnecté. Dans ce cas, *N15* contacte son prochain voisin dans la table de hachage locale *N20* pour lui demander un autre point de sortie à l'OV_{Pat}. *N20* lui communiquera l'adresse de *N19* (qui est supposé être connecté) comme point de sortie à OV_{Pat}. Ensuite, *N15* envoie un message à *N19* pour la découverte inter-*OV*. Il est important que *N15* mette à jour son point de sortie à l'OV_{Pat}. Ainsi, *N15* profite lors de l'envoi d'un message à *N19* pour la découverte inter-*OV* et lui demande l'adresse de son prochain voisin *N4*. La découverte est alors lancée dans l'OV_{Pat} à partir de *N19*. *N19* communique à *N15* son voisin *N4* comme point de sortie à l'OV_{Pat}.

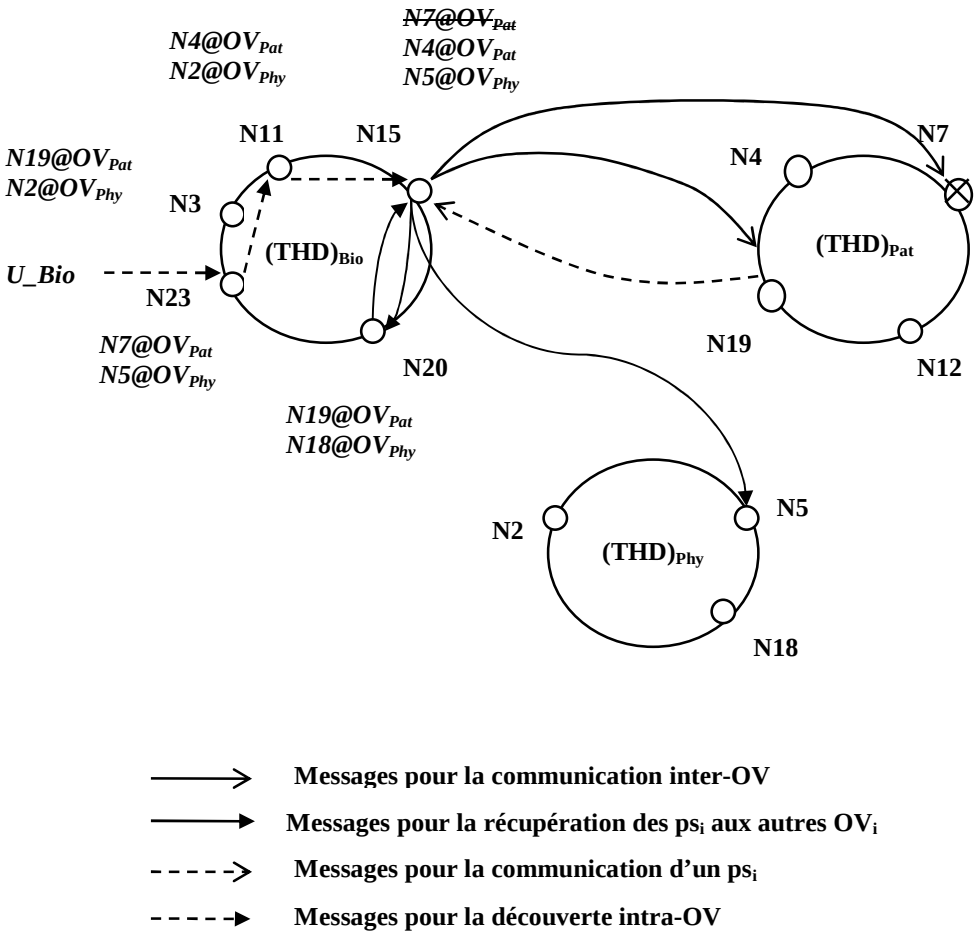

Figure 11 : Découverte inter-OV en présence d'instabilité des nœuds

3.3.6. Maintenance du système

La dynamicité des nœuds nécessite une maintenance à deux niveaux : (i) une maintenance au niveau de chaque THD_i (définie pour chaque OV_i) et (ii) une maintenance du système d'adressage. La maintenance d'une THD_i est une maintenance classique [Sto01, Row01]. La maintenance du système d'adressage consiste à définir comment les points de sortie sont établis (resp. mis à jour) lors de la connexion des nœuds (resp. lors de la déconnexion des nœuds).

Connexion d'un nœud :

Lorsqu'un nouveau nœud *Nnew* se connecte à une OV locale (OV_{locale}), la THD locale de l'OV_{locale} est mise à jour comme dans les THDs classiques [Sto01, Row01]. Ensuite, *Nnew* contacte son voisin (le plus proche) via sa table de hachage locale pour récupérer les points de sortie à toutes les autres OVs du système. Puis, *Nnew* envoie un message à chaque point de sortie psi de toute *OVi* du système en demandant l'adresse de son voisin. Si un psi ne réponds pas au bout d'une certaine période de temps (e.g un RTT), alors le psi est considéré comme déconnecté. Dans ce cas, *Nnew* contacte le voisin de son voisin dans l'OV_{locale} pour demander un nouveau psi à l'OV_i. Ceci est répété récursivement jusqu'à trouver un point de sortie connecté pour l'OV_i. Dans le cas où tous les nœuds de l'OV_{locale} sont contactés, l'OV_i est considérée comme déconnectée et nous propageons (uniquement dans ce cas) l'information sur la déconnexion de l'OV_i à tous les nœuds de l'OV_{locale}.

Comme nous avons défini une stratégie de mise à jour paresseuse, les points de sortie des voisins contactés pendant une découverte inter-OV et qui sont déconnectés ne sont pas mis à jour. La mise à jour concerne uniquement les points de sortie du nœud qui a initié le processus de la découverte inter-OV. Ainsi, nous n'enverrons pas un message aux autres OVs dans le système pour indiquer le départ d'OV_i car il est totalement inutile d'indiquer le départ d'OV_i à une OV_j ($j \neq i$) qui n'a pas eu besoin d'accéder à cette OV_i. Dans tous les cas, OV_j peut déduire la déconnexion d'OV_i lors d'une découverte inter-OV dans OV_j ou lors de l'arrivée d'un nouveau nœud dans OV_j. L'algorithme qui décrit l'établissement des points de sortie d'un nouveau nœud *Nnew* à toute OV_i dans le système est décrit ci-dessous (Figure 12).

```
// ps_i : point de sortie vers une OV_i

Récuperer tous les ps_i de chaque OV_i en contactant mon voisin ;
Pour chaque ps_i
{
      Si < ps_i n'est pas connecté>
      {
            //Cette recherche est effectuée récursivement en
            // demandant à mon voisin puis au voisin de mon
            // voisin et ainsi de suite
            Demander un nouveau ps_j ∈ OV_i au voisin de mon voisin VV
            //vérifie si on a parcouru tous les nœuds d'OV_locale
            et vérifier si VV est le nœud courant
            si c'est le cas
            {
                  //propager l'information sur la déconnexion de OV_i
                  //uniquement dans l'OV_locale
                  propager(«OV_i est déconnectée» ; OV_locale)
            } ;
            //échange de valeur entre le ps_i déconnecté et ps_j connectée
            ps_i ← ps_j ;
      }
      // associer un ps_i au nouveau nœud
      ps_i ← voisin (ps_j) ;
}
```

Figure 12 : Algorithme d'établissement des points de sortie d'un nouveau nœud *Nnew* à toute OV_i du système

3.3.7. Illustration : Etablissement des points de sortie pour un nouveau nœud

La Figure 13 illustre la connexion d'un nouveau nœud *Nnew* à l'OV_{Bio}. Une mise à jour classique de la $(THD)_{Bio}$ est effectuée [Sto01, Row01]. Il s'agit d'une mise à jour locale à l'OV_{Bio}. Ensuite, il faut établir les points de sortie pour *Nnew*. *Nnew* contacte alors son voisin *N23*. Ce dernier lui communique comme points de sortie *N19* à l'OV_{Pat} (resp. *N2* à l'OV_{Phy}). *Nnew* contacte *N19* (resp. *N2*) en lui demandant l'adresse de son voisin. Enfin, *N19* (resp. *N2*) communique l'adresse de son prochain voisin dans sa table de hachage locale *N4* (resp. *N5*) à *Nnew* via sa table de hachage locale.

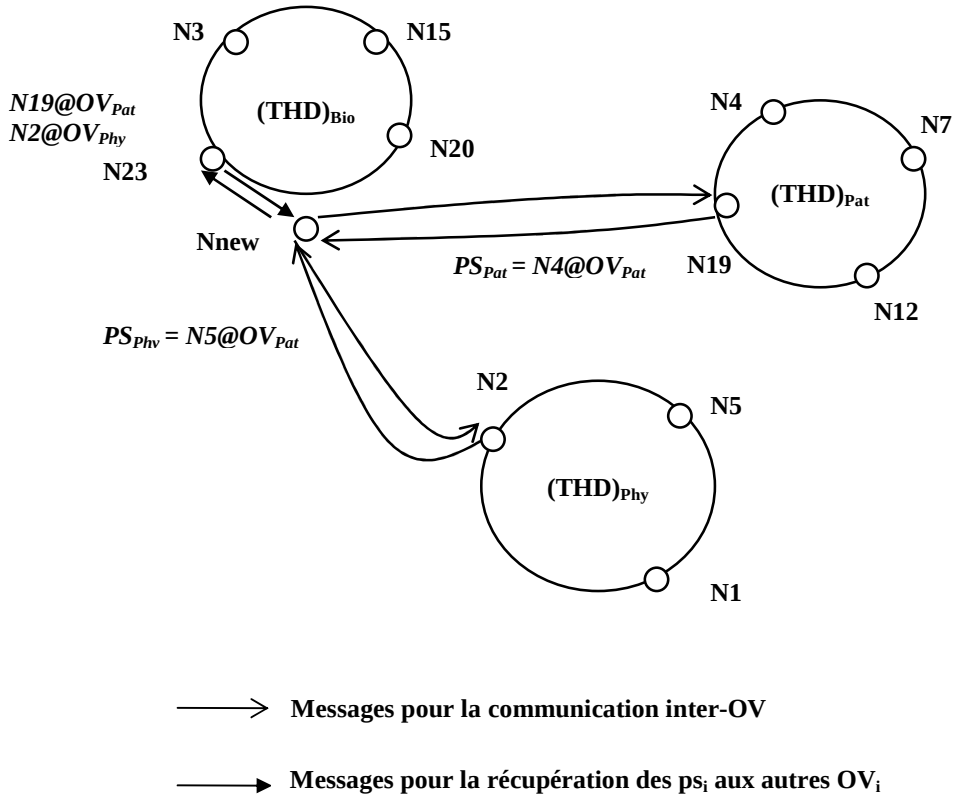

Figure 13 : Etablissement des points de sortie pour *Nnew* en absence de la dynamicité des nœuds

Dans l'exemple précédent, les points de sortie *N19* et *N2* sont supposés être connectés respectivement à l'OV_{Pat} et à l'OV_{Phy}. Supposons maintenant que le point de sortie *N19* soit déconnecté (Figure 14). Dans un tel cas, *Nnew* contacte le voisin de son voisin *N3* qui lui communique *N7*. Ensuite, *Nnew* contacte *N7* (qui est bien connecté) pour récupérer l'adresse de son voisin *N12*. Enfin, *Nnew* peut mémoriser *N12* comme point de sortie à l'OV_{Pat}. Le point de sortie à l'OV_{Pat} du nœud *N23* ne sera pas mis à jour car nous utilisons une stratégie de mise à jour paresseuse.

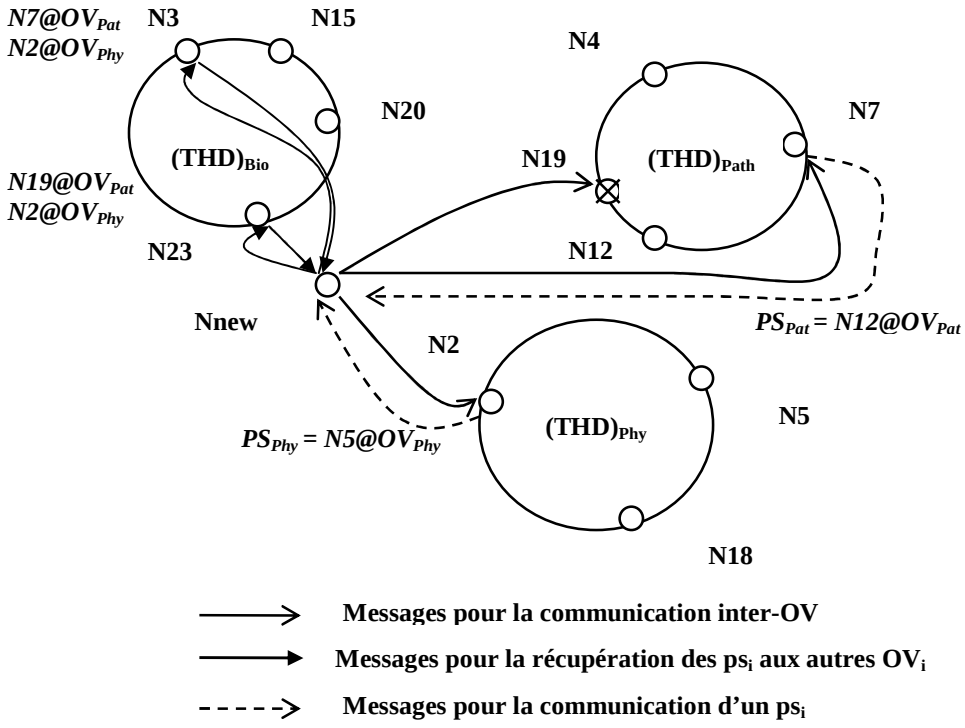

Figure 14 : Etablissement des points de sortie pour *Nnew* en présence de la dynamicité des nœuds

Déconnexion d'un nœud :

Lorsqu'un nœud *NDisc* se déconnecte d'une *OV*, une mise à jour classique pour la THD locale est effectuée. Ensuite, nous avons deux possibilités pour la maintenance du système d'adressage: soit *NDisc* on effectue une mise à jour immédiate en propageant l'information de sa déconnexion à toutes les *OV* du système, soit *NDisc* n'indique pas son départ.

Dans le premier cas, *NDisc* doit propager l'information à toutes les autres OV_i pour indiquer son départ en utilisant ses points de sortie ps_i. Ensuite, le ps_i doit informer tous les nœuds de son OV_i du départ de *NDisc*. Une inondation sera générée dans chaque OV_i du système. Cette solution amène clairement à des mauvaises performances.

Dans le deuxième cas, aucun message n'est envoyé pour indiquer le départ de *NDisc*. Nous choisissons plutôt cette stratégie car l'algorithme de découverte des métadonnées prend en considération la dynamicité des nœuds (Figure 9). La mise à jour d'un point de sortie (qui est

déconnecté) est effectuée uniquement lors de la découverte inter-OV ce qui garantit un accès permanent d'une OV_{locale} à n'importe quelle OV_i ($i \neq locale$).

4. Conclusion

Dans ce chapitre, nous avons proposé, en premier lieu, une architecture pour l'évaluation de requêtes en environnement de grille de données. Nous avons situé les différentes phases d'évaluation de requêtes dans une telle architecture. Nous avons aussi montré les différentes interactions entre ces phases et les deux services : le Service de Découverte de Ressources (SDR) et le service de monitoring de ressources de calcul, le NDS.

Les approches traditionnelles (i.e. centralisée et répliquée) pour l'organisation du schéma global ne peuvent pas passer à l'échelle [Pac07]. Dans ce contexte, nous proposons le SDR qui permet une découverte efficace de sources de données. Ce service définit aussi une stratégie simple pour la découverte de ressources de calcul. Les ressources découvertes (e.g. relations, CPU d'un nœud) doivent être observées vu que ces dernières sont partagées à grande échelle. D'où, la nécessité de l'utilisation d'un service de monitoring de ressources pour observer l'état actuel des ressources. Le but de cette observation (i.e. du monitoring) est de réagir en fonction des erreurs d'estimation, et des variations de valeurs de paramètres hôtes et réseaux en prenant des décisions. Par exemple, si l'optimiseur décide d'exécuter une opération sur un nœud, ce choix peut changer en fonction de l'état actuel du nœud durant l'exécution. Dans ce contexte, les méthodes d'optimisation dynamique permettent d'adapter les plans d'exécution aux changements de paramètres (e.g. hôtes) de l'environnement d'exécution.

Ensuite, nous avons proposé une méthode de découverte de sources de données pour l'évaluation de requêtes en environnement de grille de données. La méthode proposée consiste à définir une THD_i pour chaque OV_i dans le système. L'interconnexion entre les différentes (OV_i)s est établit grâce à un système d'adressage permettant un accès permanent de n'importe quelle OV_{locale} à toute autre OV_i ($i \neq locale$). Le système d'adressage est maintenu lors de la découverte inter-OV. Ce système emploie une mise à jour paresseuse évitant ainsi un surcoût de maintenance en présence de l'effet churn. Les avantages de notre proposition sont : (i) un meilleur partage de charge au sein d'une même OV_{locale} car les points de sortie vers une OV_i ($i \neq locale$) ne sont pas tous identiques, (ii) une découverte inter-OV robuste (en présence de la dynamicité des nœuds) car un nœud peut contacter plusieurs voisins d'une manière récursive jusqu'à trouver un point de

sortie connecté, (iii) éviter la convergence, dans le cas d'un effet churn, de toutes les requêtes adressées à une OV_i vers un seul point de sortie et (iv) un faible coût de maintenance en présence entre les THD_i (où chaque THD_i est associée à une OVi) en présence de l'effet churn.

Le Tableau 3 compare les méthodes basées sur des techniques P2P avec notre méthode [Sam09], par rapport aux critères déjà présentées dans le chapitre précédent.

Le Tableau 4 compare les coûts de découverte de ressources des méthodes basées sur des techniques P2P par rapport à notre méthode [Sam09]. Notre méthode de découverte de ressource peut générer un nombre plus important de messages si la découverte dans une OV_{locale} échoue mais elle possède les avantages de la prise en compte de principe de la localité de données. D'autre part, elle minimise le coût de la maintenance des THDs.

La méthode proposée peut être étendue pour la découverte de n'importe quel type de ressource (e.g. un programme distant, un service web) identifié par un nom (e.g. «programme α», « service web β»). En effet, dans un système de grille les utilisateurs d'une OV peuvent avoir besoin à accéder à des ressources qui n'appartiennent pas à leur OV. Par exemple, un physicien d'une OV_{Phy} peut avoir besoin à accéder à un programme de calcul matriciel (e.g. « programme φ ») appartenant à une OV_{Maths}.

Dans le chapitre suivant, nous présentons une étude sur le monitoring de ressources de calcul pour l'optimisation dynamique de requêtes en environnement de grille de données. Nous étudions l'apport d'un service de monitoring pour un modèle d'exécution à base d'agents mobiles [Arc04, Hus05b, Mor03]. Dans ce contexte, nous proposons une méthode permettant la prise en considération de la variation des valeurs des paramètres hôtes (e.g. valeur de la charge CPU d'un nœud) et réseaux (e.g. valeur de la charge de la bande passante entre deux nœuds) dans le calcul du coût d'exécution d'une opération relationnelle. La méthode proposée est basée sur l'utilisation des informations de monitoring pour le calcul des valeurs des paramètres hôtes et réseaux à jour, pendant l'exécution.

Critères / Catégorie	Sous-catégorie	Passage à l'échelle	Instabilité	Propagation de la découverte				Maintenance élevée (en présence du churn)	Découverte fiable	Requêtes complexes	
				Succession d'unicasr	Anycast	Multicast	broadcast			Statique	dynamique
P2P	Non-structuré		X				X		X	X	X
	Super-pair					X			X	X	X
	Structuré (THD)	X	X	X				X	X	X	
	plusieurs THDs [Sam09]	X	X	X					X	X	

Tableau 3 Comparaison des méthodes P2P par rapport à notre méthode [Sam09]

Critères / Catégorie	Sous-catégorie	Coût de la découverte	
		Temps de réponse	Nombre de messages
P2P	Non-structuré	$O(d)$	$O(N^2)$
	Super-pair	$O(S+\Sigma C_{pi(1 \le i \le S)})$	$O(S^2)$
	Structuré (THD)	$O(Log(N))$	$O(Log(N))$
	plusieursTHD [Sam09]	$O(Log(N))$	$(K\text{-}1) * O(Log(M_i)) + \sum coût\_accès(ps_i)_{(1 \le i \le k-1)}$ (avec M le nombre moyen de nœuds dans une OV et K le nombre d'OVs dans le système).

Tableau 4 Comparaison des coûts de découverte de ressources des méthodes P2P par rapport à notre méthode [Sam09]

Chapitre IV. Monitoring de ressources de calcul pour l'optimisation dynamique de requêtes dans une grille de données

1. Introduction

Les plans d'exécution engendrés par les optimiseurs traditionnels peuvent être sous-optimaux à cause de [Hus05a] : (i) la centralisation des décisions prises par l'optimiseur, (ii) l'inexactitude des estimations (e.g. l'estimation de la taille des données) et (iii) l'absence des valeurs des paramètres à jour, hôtes (i.e. les valeurs de la charge CPU, de la charge des E/S et de la charge mémoire d'un nœud) et réseaux (i.e. les valeurs de la latence et de la charge de la bande passante entre deux nœuds). Les problèmes d'optimisation dus aux erreurs d'estimation et à l'indisponibilité des données ont été largement étudiés dans les systèmes parallèles et distribués. Les méthodes d'optimisation dynamique peuvent être classées principalement en deux catégories [Hus06]: les méthodes d'optimisation dynamiques centralisées [Ams98, Avn00, Bou00, Ive04, Kab98, Zho05] et les méthodes d'optimisation dynamiques décentralisées [Col04, Hus05b, Jon97, Mor03, Urh00, Ive99].

Dans les méthodes d'optimisation dynamique centralisées, le contrôle et les modifications d'un plan d'exécution sont gérés par un optimiseur centralisé. Ce dernier contrôle tous les processus participant à l'optimisation. La centralisation des décisions prises par l'optimiseur peut créer un goulet d'étranglement dans un système à grande échelle à cause du nombre important de messages échangés dans un réseau à faible débit et à forte latence. Pour éliminer ce goulet d'étranglement, les méthodes décentralisées ont été proposées. Les méthodes décentralisées décrites dans [Col04, Urh00, Ive99] améliorent le coût du traitement local en adaptant l'utilisation des ressources CPU, E/S et mémoire en présence des erreurs d'estimation. Dans le but de considérer, en plus des ressources ci-dessus, les ressources réseaux, les méthodes de

[Hus05a, Jon97, Mor03] on été proposées. Ces méthodes sont basées sur l'utilisation des agents mobiles pour corriger la sous-optimalité des plans d'exécution en décentralisant le contrôle et les modifications de ces plans.

Dans ce chapitre, nous reprenons les travaux de [Arc04, Ham06, Hus05a, Mor03]. Ces derniers proposent un modèle d'exécution à base d'agents mobiles pour l'optimisation dynamique de requêtes réparties à grande échelle. Dans ce modèle, chaque opérateur relationnel (nommé opérateur mobile) est exécuté au moyen d'un agent mobile. Un agent mobile réagit, en cours d'exécution, aux erreurs d'estimation et à l'indisponibilité des ressources en migrant (éventuellement) d'un nœud à un autre. Ainsi, pour qu'un agent puisse migrer d'une manière autonome et sans être dépendant de l'optimiseur, il a besoin d'une politique de migration proactive qui lui permet de choisir son nœud d'exécution le plus adéquat. Cette politique de migration doit tenir compte non seulement des erreurs d'estimation mais aussi de l'indisponibilité des ressources sollicitées (e.g. données, CPU). Pour mettre en œuvre cette politique de migration proactive, nous nous appuyons sur un modèle de coûts. Ce dernier est constitué de deux parties : la partie résidente sur un nœud et la partie embarquée dans un agent. Le modèle de coûts embarqué [Hus05b] permet à un agent d'assurer son autonomie en évitant des interactions distantes avec un optimiseur et en les remplaçants par des interactions locales.

Le nœud de migration d'un agent est choisi en fonction de plusieurs métriques : (i) le coût de production locale des opérandes, (ii) le coût d'exécution local d'un agent, (iii) le coût de migration d'un agent, (iv) le coût de transfert des opérandes et (v) le coût de transfert du résultat. Ces différentes métriques sont calculées en fonction du profil des opérandes et du résultat. Les métriques (i) et (ii) dépendent des valeurs des paramètres hôtes (e.g. valeur de la charge CPU) des nœuds stockant les relations référencées dans une requête. Les métriques (iii), (iv) et (v) sont calculées en fonction des valeurs des paramètres réseaux (e.g. valeur de la charge de la bande passante) entre les nœuds stockant les relations.

Dans la littérature, plusieurs approches ont été proposées pour la définition des modèles de coûts permettant l'estimation du coût des opérations : l'approche historique [Ada96], l'approche par calibration [Du95, Gar96] et l'approche générique [Naa99][18]. Dans ces approches, le modèle de coûts utilise des valeurs pour des paramètres hôtes et réseaux qui peuvent souvent être obsolètes ou imprécises. Étant donné le grand nombre de ressources partagées dans une grille, les

[18] Nous détaillons dans la section suivante le principe de chaque approche

paramètres à jour reflétant l'état actuel de ces ressources sont indispensables afin de mieux estimer le coût de traitement d'une opération. Ces paramètres nécessitent un outil de monitoring pour observer l'état de ressources. Ainsi, il devient primordial de permettre au modèle de coûts d'accéder aux paramètres, à jour, hôtes et réseaux dans un système de grille. Dans ce contexte, nous proposons une méthode permettant la prise en compte de la variation des valeurs des paramètres hôtes et réseaux dans le calcul du temps d'exécution d'une opération relationnelle. La méthode proposée est basée sur l'utilisation des informations de monitoring pour le calcul des valeurs des paramètres hôtes et réseaux à jour, pendant l'exécution. D'une manière plus précise, un agent s'adresse au modèle de coûts en lui demandant d'estimer le coût d'exécution *CE* d'une ou plusieurs opérations (e.g. l'étape de sondage) sur son espace de migration (i.e. l'ensemble de nœuds où un agent peut migrer). Afin de mieux estimer *CE*, l'agent fournit des informations très utiles (qui figurent dans le modèle de coûts embarqué et des valeurs pour des paramètres hôtes et réseaux à jour calculées via un service de monitoring, le NDS) au modèle de coûts pour mieux estimer *CE*. Ensuite, l'agent peut prendre une décision efficace en fonction de l'estimation du *CE* sur les différents nœuds de l'espace de migration. La méthode proposée améliore nettement la qualité de décision de l'optimiseur [Sam08].

Le reste de ce chapitre est organisé comme suit : dans la section 2, nous décrivons d'une manière plus précise le problème évoqué dans ce chapitre. La section 3 présente quelques services de monitoring utilisés dans les systèmes de grille d'aujourd'hui suivi des scénarios d'utilisation des services de monitoring dans le contexte de traitement de requêtes dans les systèmes de grille. Ensuite, nous justifions le choix de l'outil de monitoring utilisé, le NDS. La section 4 décrit le modèle de coûts. La section 5 explique comment les informations de monitoring (calculées par NDS) sont intégrées dans une jointure mobile. Enfin, nous concluons dans la section 6.

2. Contexte

2.1. Jointure mobile par hachage

Considérons une jointure *J* : $T = R1 \infty R2$ avec deux relations *R1* et *R2* qui résident respectivement sur deux nœuds *N1* et *N2*. Nous supposons que la taille de *R1* est plus petite que la taille de *R2* (i.e. $|R1| < |R2|$). La jointure classique par hachage simple [Ozs99] s'exécute en deux

étapes : (E1) une table de hachage *TH* est construite avec la plus petite (i.e. taille) des deux relations (dans notre exemple *R1*) et (E2) la table de hachage est sondée avec l'autre relation *R2*.

Durant la construction de la table de hachage, le profil de *R1* (e.g. taille de *R1*) peut être calculé précisément. En considérant le profil précis de *R1*, le profil de *R2* estimé à la compilation et la taille révisée de *T* à partir de *R1* et *R2*, l'indisponibilité de *R2* (i.e. c'est le temps d'attente pour recevoir *R2*) et les caractéristiques de ressources (e.g. temps CPU pour comparer un tuple, valeur de la charge CPU, valeur de la charge de la bande passante entre *N1* et *N2*), il est possible de prendre une décision sur la localisation de l'étape de sondage et éventuellement de déplacer cette étape sur un autre nœud (on se limitera soit à *N1*, soit à *N2*). L'algorithme de jointure par hachage simple est étendu en permettant à l'agent de choisir le nœud d'exécution de la jointure entre la phase de hachage et celle du sondage [Arc04].

La jointure mobile n'a pas été développée dans un environnement de grille de données qui se caractérise par la (i) grande échelle et (ii) l'instabilité. Pour la prise en compte de la caractéristique de la grande échelle (i) (e.g. plusieurs utilisateurs peuvent se connecter à un même nœud pour lancer des requêtes SQL par exemple), il est très important d'observer les valeurs des paramètres influençant le comportement de la jointure mobile. En particulier, nous nous intéressons, dans cette étude, à l'impact de la variation des valeurs des paramètres hôtes et réseaux sur le comportement de la jointure mobile. Quant à la prise en compte de la caractéristique de l'instabilité (ii), nous supposons que les nœuds *N1* et *N2* sont bien connectés durant l'évaluation de la jointure mobile. Dans le cas contraire (i.e. si un des nœuds *N1* ou *N2*, n'est pas connecté au système), l'évaluation de la jointure mobile ne peut pas s'effectuer car les deux relations *R1* et *R2* sont indispensables pour l'exécution de la jointure mobile.

Dans la suite de ce chapitre, les coûts nécessaires pour le choix du nœud d'exécution d'un agent mobile sont aussi nommés les métriques d'agent. Le comportement d'un agent mobile, exécutant une jointure mobile par hachage simple, est décrit dans l'algorithme de la Figure 15 [Arc04]. Dans la section suivante, nous présentons d'une manière plus précise le problème abordé dans ce chapitre.

```
Soient
        R1 et R2 deux relations,
        TH la table de hachage de R1,
        T le résultat de jointure entre R1 et R2,
        N le noeud d'exécution

Si (non local (R1) ) Alors Recevoir (R1) ;
Construire (R1, TH);
N ← decision (ProfRel, IndRel, EtatRes) ;
Si  (non local (N)) Alors Migrer sur N ;
Si (non local (R2) )  Alors Recevoir (R2) ;
Sondage (TH, R2, T) ;
Si (non local (T)) Alors Envoyer (T)
Sinon Materialiser (T) ;
```

Figure 15 : Algorithme de la jointure mobile par hachage

2.2. Position du problème

La décision de migration d'un agent est prise en fonction du résultat de la comparaison du coût d'exécution d'une partie de la jointure *J* sur *N1* et sur *N2*. Plus précisément, si le coût de traitement de l'étape de sondage sur *N1* et l'envoi de *R2* (de *N2* à *N1*) sont inférieurs au coût de l'envoi de la table de hachage de *R1* (de *N1* à *N2*) plus le coût de traitement de l'étape de sondage sur *N2* ainsi que le coût de l'envoi de résultat T (de *N2* à *N1*) alors l'agent décide d'exécuter *J* sur *N1*. Sinon il décide d'exécuter *J* sur *N2*.

L'agent mobile demande au nœud où il se trouve (i.e. *N1*) d'estimer ces différents coûts. Comme nous l'avons précisé en introduction, le modèle de coûts peut suivre plusieurs approches : l'approche par calibration [Du95, Gar96], l'approche historique [Ada96] et l'approche générique [Naa99]. L'approche par calibration [Du95, Gar96] se base sur les statistiques décrivant les données issues d'une source pour déduire les formules de coûts estimant les coûts des sous-requêtes exécutées par les sources. Elle suppose que toutes les sources de données aient un modèle de coûts identique. Dans cette approche, les coefficients des formules de coûts sont estimés en exécutant des requêtes de calibration [Du95]. La procédure de calibration est effectuée une seule fois pour chaque source de données. La calibration peut être efficace pour certaines sources de même type, telles que les SGBD relationnelles. Cependant, il est très difficile de calibrer des sources de données très diversifiées avec le même modèle de coûts. De plus, les valeurs des paramètres hôtes et réseaux calibrés peuvent changer de valeur entre deux instants. L'approche historique [Ada96] consiste à estimer les coûts des sous-requêtes traitées par les sources en s'appuyant sur les exécutions antérieures. Cette approche maintient dans la base

historique d'un nœud des informations sur les sous-requêtes exécutées par les sources. Elle associe ainsi, à chaque sous-requête qui a été exécutée par une source, un historique. Cette approche est efficace pour l'estimation des coûts des requêtes similaires. Elle suppose que la valeur des paramètres hôtes et réseaux soient identiques entre deux exécutions antérieures. Cependant, ces paramètres peuvent changer de valeur d'une manière significative. De plus, pour les requêtes soumises la première fois, leur coût est estimé aveuglément par le système. [Naa99] propose un modèle de coûts intégrant le modèle de coûts initial d'un médiateur et les modèles de coûts associés aux sources. Le modèle de coûts initial d'un nœud sert à estimer : (i) les coûts des opérations traitées par le nœud, et (ii) les coûts des sous-requêtes exécutés par les sources qui ne divulguent pas leurs modèles de coûts. Ceux-ci sont estimés par des formules de coûts génériques qui peuvent être déduites par l'approche de calibration ou par l'approche historique. Les trois approches ci-dessus ne prennent pas en considération la variation des valeurs des paramètres hôtes et réseaux dans le temps. Or, la valeur de ces paramètres (e.g. valeur de la charge CPU d'un nœud, valeur de la charge de la bande entre deux nœuds) peut changer d'une manière significative entre deux instants, dans un système de grille. Le problème est alors de comment fournir au modèle de coûts la meilleure valeur estimée d'un paramètre hôte ou réseau. Nous proposons d'utiliser les informations de monitoring pour le calcul des paramètres hôtes et réseaux pendant l'exécution. Ces paramètres sont intégrés dans la formule de coûts estimant le coût d'exécution d'une partie de la jointure mobile. Ainsi, notre choix permet de prendre en compte la variation de valeurs des paramètres hôtes et réseaux dans le calcul du temps de réponse d'une jointure mobile. Dans la section suivante, nous justifions notre choix pour l'utilisation d'un service de monitoring, bien adapté à notre contexte.

3. Choix d'un service de monitoring

Le but de cette section est de présenter les outils de monitoring les plus utilisés et de nous guider dans l'utilisation d'un service de monitoring adapté à notre contexte. Avant de présenter les services (ou les outils) de monitoring les plus connus, nous présentons d'abord l'architecture considérée aujourd'hui comme la référence pour un outil de monitoring dans un système de grille, le Grid Monitoring Architecture (GMA) [Tie02]. Ensuite, nous présentons les services de monitoring les plus utilisés suivi des scénarios d'utilisation du monitoring pour le traitement de

requêtes dans un système de grille. Enfin, nous justifions clairement notre choix de service de monitoring : le Network Distant Service (NDS).

3.1. Une architecture de monitoring en grille [Tie02]

Dans un but d'unifier l'architecture de monitoring d'un système de grille, le "Network Metric Working Group" du "Global Grid Forum (GGF)" une architecture de monitoring en grille, le "Grid Monitoring Architecture" (GMA). Le GMA décrit les principaux composants d'une architecture de monitoring et les interactions entre ces composants. GMA peut être vu comme un modèle ou une spécification dont le but est d'aider et de guider les concepteurs des outils de monitoring dans un système de grille. Elle inclut trois types de composants (Figure 16):

- Les producteurs déclarent la présence des nouvelles ressources en fournissant des informations (e.g. adresse d'une ressource) auprès de l'annuaire afin que ces ressources soient visibles au niveau du système.
- Les consommateurs envoient des requêtes pour récupérer l'information concernant une ressource. Un consommateur peut être un utilisateur ou une application.
- Les annuaires permettent la publication et la recherche des ressources. L'annuaire peut être vu comme un intermédiaire permettant la communication entre les producteurs et les consommateurs. Il est souvent organisé suivant une approche centralisée (e.g. [And04, Gos07a, Val08]) ou hiérarchique (e.g.[Coo03, Sch06, Wol99]).

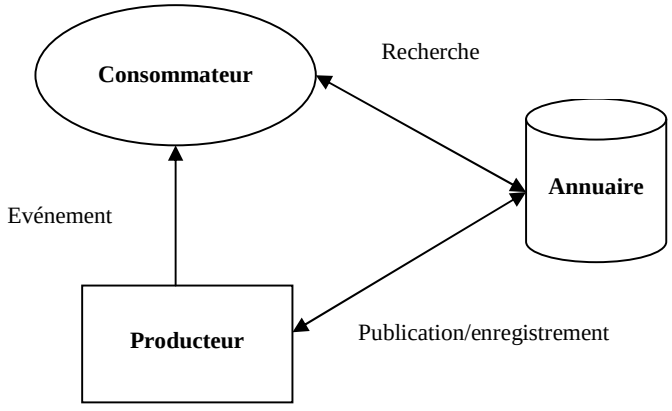

Figure 16 : Architecture GMA

Le fonctionnement du GMA est le suivant. D'abord, les producteurs déclarent la présence de nouvelles ressources {*R1, R2, …, Rn}* et fournissent l'information nécessaire pour les localiser en s'inscrivant dans l'annuaire. Si un consommateur *C* a besoin d'une ressource *Ri* alors *C* enverra une requête à l'annuaire qui, à son tour, lui communiquera la référence (e.g. l'adresse IP) de *Ri*. Un lien est directement formé entre le producteur de *Ri* et *C* pour s'échanger des paramètres (e.g. charge CPU). Le producteur est responsable de communiquer les événements via une ou plusieurs ressources. Une ressource peut être un capteur matériel (« hardware ») ou logiciel (« software ») ou un système de monitoring comme le Network Weather Service (NWS) [Wol99]. Le principal avantage de l'architecture GMA est de séparer la tâche de la recherche de ressources et la tâche de l'échange de données entre un producteur et un consommateur. Enfin, GMA peut être vu comme un modèle qui reste très général, e.g. GMA ne précise pas le langage de requêtes ou le protocole de communication pour échanger les messages entre les différents composants [Coo03, Bad07].

3.2. Services de monitoring

Les outils compatibles avec GMA diffèrent principalement selon le type de ressources qu'ils observent. Par exemple, le Monitoring and Discovery Service (MDS) [Fos05] et le Relational-GMA (R-GMA) [Coo03] permettent d'effectuer des mesures sur des ressources de calcul (plus précisément l'observation des paramètres hôtes) tandis que l'outil SCALEA-G [Tru04] permet le monitoring des ressources logicielles. D'ailleurs, ces outils utilisent des formats de stockage et des langages de requêtes différents : R-GMA utilise une base de données virtuelle qui peut être interrogé avec un langage de requêtes (e.g. SQL) et MDS utilise des fichiers en format XML. Une autre référence est le Network Weather Service (NWS) [Wol99]. Ce service permet d'observer des ressources de calcul (observation des paramètres hôtes et réseaux). Une étude détaillée sur les outils de monitoring de ressources de calcul pour des systèmes de grille figure dans [Zan05]. Nous présentons aussi sous forme d'une grille une quinzaine de travaux sur outils de monitoring pour des systèmes de grille dans l'annexe.

3.3. Services de monitoring pour le traitement de requêtes dans un système de grille

Dans un contexte de traitement de requêtes distribuées classique, les valeurs des paramètres hôtes et réseaux à jour sont absentes, imprécises ou incomplètes [Gou04a]. En effet, toutes les ressources sont souvent sous un contrôle central dans les systèmes de base de données parallèles

et distribués. Par contre, un système de grille diffère principalement par le partage de ressources entre plusieurs OVs. Il est important en particulier de surveiller les ressources de calculs afin d'exécuter efficacement des requêtes dans un système à grande échelle. Dans cette sous-section, nous présentons plusieurs scénarios d'utilisation d'outils de monitoring pour le traitement de requêtes dans un système de grille. En effet, les informations du monitoring peuvent être utilisées pendant la phase de compilation [Gou06, Smi02] ou pendant la phase d'exécution [Gou04a, Sam08, Woh07].

3.3.1. Monitoring pendant la phase de compilation

Les informations de monitoring sont utilisées pendant la compilation pour un placement efficace des opérateurs relationnels dans [Alo05, Gou06, Smi02, Lyn09]. C'est un scénario classique d'utilisation des informations de monitoring où l'optimiseur a besoin de récupérer des paramètres (hôtes et réseaux) à jour afin de prendre des décisions (e.g. les nœuds ayant leur E/S non chargés sont utilisés pour l'extraction des données [Gou06]). [Liu08] propose une stratégie de sélection de ressources pour l'optimisation de requêtes dans un système de grille. La sélection de ressources est effectuée avec une fonction qui permet de calculer le coût d'exécution d'une requête pour un nœud donné. Cette fonction utilise trois services d'informations qui permettent de capturer l'état de l'environnement : Environment Information Service (EIS), Transmission Predection Service (TPS) et Database Information Service (DIS). L'EIS est responsable de fournir des paramètres hôtes tandis que le TPS permet de fournir des paramètres réseaux. Enfin, le DIS gère un catalogue qui maintient le profil des données.

3.3.2. Monitoring pendant la phase d'exécution

[Gou04a] propose un Framework de monitoring de ressources pendant l'exécution pour un traitement de requêtes dynamique dans un système de grille. Nous rappelons qu'un traitement de requêtes est définit comme étant dynamique [Hel00] s'il reçoit l'information de son environnement et détermine son comportement d'une manière itérative. Le plan d'exécution (PE) et l'environnement d'exécution sont observés. Ensuite, une évaluation est effectuée en fonction des informations observées. Enfin, une réponse est envoyée à l'évaluateur de requêtes pour prendre une décision. Ceci peut affecter l'évaluation de la requête en cours. La réponse peut se limiter à la redirection du prochain tuple à un nœud particulier ou la réexécution de l'optimiseur sur une partie de la requête voire toute la requête. [Gou04a] identifie trois phases:

- **Le monitoring** reçoit l'information -à jour- sur les ressources inscrites dans l'évaluation d'une requête et génère des événements en fonction des informations reçues.
- **L'estimation** a pour fonction d'évaluer les événements reçus par le monitoring et de vérifier s'il faut changer des valeurs pour des propriétés intéressantes afin d'optimiser la requête en cours d'exécution. L'approche utilisée le plus souvent dans cette phase est Evénement-Condition-Action.
- **La réponse** s'occupe d'envoyer des réponses à l'évaluateur de requêtes afin de lui notifier des modifications éventuelles. Cette phase se fait dans le cas où l'estimateur a pu identifier des changements. Le système essaie alors d'identifier des méthodes efficaces pour répondre aux besoins. Le moteur d'exécution est ainsi notifié et les modifications auront lieu lors de la phase réponse

Dans [Gou04b], les auteurs proposent une méthode pour l'auto-monitoring du plan d'exécution (PE) d'une requête. Plus précisément, l'auto-monitoring est effectué sur les opérateurs algébriques. Par exemple, le coût estimé d'une jointure par hachage pendant la compilation utilise une valeur constante pour le temps de hachage d'un tuple. Cette constante peut être observée puis corrigée si elle n'est pas précise. Une autre constante peut aussi être observée est le temps de concaténation de deux tuples (qui est souvent difficile à estimer). Ces informations observées sont utilisées par l'optimiseur pendant l'exécution et le PE peut être modifié en fonction de ces informations. Ce type d'information peut aussi aider pour la calibration du modèle de coûts afin d'améliorer la précision de ces paramètres. [Woh07] présente un service de monitoring capable de fournir des paramètres concernant des sources de données pendant l'exécution dans un système de grille. Par exemple, ces paramètres peuvent aider à la prise de décisions pour changer la source de données actuelle (qui ne dispose pas d'index pour cette requête) d'une requête vers une copie de la source de données (index utilisable pour cette requête) afin d'améliorer les performances de cette requête.

3.4. Service de monitoring pour des agents mobiles

Les agents mobiles doivent se déplacer d'une manière simple et efficace d'un nœud à un autre. Ils peuvent aussi changer d'OV pendant leurs déplacements. Par conséquence, ils peuvent aussi changer d'outils de monitoring. Les outils de monitoring différent naturellement par leur

langage de requêtes. Embarquer plusieurs interfaces pour l'invocation des outils de monitoring dans un agent mobile est très dur pour des raisons liées à la taille d'un agent et à la complexité de la programmation de plusieurs méthodes dans un agent. En effet, ceci peut rapidement devenir complexe et très coûteux surtout si le nombre d'outils utilisés est très grand et si ces outils différent d'une OV à une autre (ce qui est tout à fait normal dans des OVs à grande échelle). Par conséquence, il est plus performant d'embarquer une seule interface, entre les agents mobiles, capable d'estimer la valeur d'un paramètre utilisé pour un agent. Dans ce contexte, nous proposons d'utiliser NDS [Gos07a, Gos07b].

3.4.1. NDS

NDS est une interface unique pour tous les outils de monitoring dans le système. NDS est capable de fournir la valeur des paramètres hôtes et réseaux à jour, indispensables pour la prise de décision d'un agent. Ainsi, les agents mobiles peuvent se déplacer d'un nœud à un autre avec un minimum de code vu qu'ils n'ont à embarquer qu'une seule méthode d'invocation de NDS. Ce dernier a un double objectif. D'abord, c'est une interface unique pour tous les outils de monitoring. D'autre part, il convertit tous types de paramètres sous un seul format prédéfini quelque soit l'outil de monitoring que NDS utilise. Il permet également de calculer la meilleure valeur estimée pour un paramètre si celle-ci est fournie par plusieurs outils de monitoring. Ainsi, les agents mobiles doivent embarquer une seule interface pour l'utilisation de NDS et ils n'ont pas besoin de s'occuper des détails concernant la récupération ou la conversion des informations de monitoring de n'importe quel type d'outil. En effet, les agents mobiles peuvent invoquer NDS pour récupérer la meilleure valeur estimée d'un paramètre hôte ou réseau, pendant l'exécution. Un agent peut décider de récupérer un ou plusieurs paramètres en fonction des paramètres nécessaires pour une opération (e.g. pour l'étape de sondage, il est surtout intéressant de récupérer la valeur de la charge CPU ; quant à l'envoi d'une relation entre deux nœuds il est intéressant de récupérer la valeur de la charge de la bande passante). Les paramètre hôtes sont la charge CPU, la charge des E/S et la charge mémoire d'un nœud. Les paramètres réseaux sont la latence et la bande passante entre deux nœuds.

Le NDS est conçu pour utiliser facilement les mesures fournies par n'importe quel outil de monitoring. En effet, les outils de monitoring sont accessibles grâce à des simples lignes de commande d'exécution. Aucun médiateur ne doit alors être développé et l'intégration des

nouveaux paramètres et outils est fait par un fichier de configuration Java de type Java Naming and Directory Interface (JNDI) dans NDS. La seule condition pour cette intégration est que les outils de monitoring soient interrogeables depuis l'hôte d'exécution du NDS. En plus, NDS est compatible avec OGSA, ce qui permet une intégration facile avec des applications tournant sur Globus.

4. Modèle de coûts

Le modèle de coûts permet de calculer un ensemble de métriques exploitées par l'optimiseur afin de générer un plan d'exécution optimal ou proche de l'optimal [Ham96]. Ces métriques vont aider l'optimiseur à choisir l'algorithme adéquat pour chaque opérateur, la meilleure stratégie de recherche et un ordonnancement efficace des jointures.

Le modèle de coûts est constitué de deux composants : (i) les métriques et (ii) les bibliothèques. Les métriques (i) regroupent les services rendus à l'optimiseur. Par exemple, une métrique qui nous intéresse particulièrement est l'estimation du coût de traitement de l'étape de sondage (une métrique peut aussi être le coût de traitement d'une jointure mobile ou bien d'une partie d'une jointure mobile). Les bibliothèques (ii) contiennent des métadonnées décrivant les relations référencées dans une requête et les nœuds (ici c'est *N1* et *N2*) intervenant dans le plan d'exécution et les formules de coûts. Les métadonnées décrivant les nœuds contiennent uniquement des métadonnées statiques [Sam07] comme nous l'avons indiqué dans le chapitre précédent. Ces métadonnées statiques contiennent pour un nœud donné : le temps E/S (i.e. le temps pour lire une page, le temps pour écrire une page), le temps CPU (i.e. le temps pour effectuer une opération simple, e.g. temps de comparaison de deux attributs), la taille mémoire en octets, (i.e. la taille d'une page |P|). L'ensemble de ces métadonnées (décrivant les relations référencées dans une requête et les nœuds intervenant dans le plan d'exécution) est récupéré pendant la phase de découverte de ressources de calcul (voir la chapitre précédent, architecture d'évaluateur de requêtes SQL en environnement de grille de données). Le profil des relations est comparé avec le profil qui figure dans le nœud local (nous supposons que le profil d'une donnée est toujours estampillé par une date). L'estampillage permet de comparer les métadonnées récupérées pendant la phase de découverte de sources de données avec les métadonnées stockées dans un nœud local afin d'utiliser les plus récentes. Les formules de coûts (ii) peuvent être répliqués sur tous les nœuds car ils ne sont pas mis à jour fréquemment. Enfin, les valeurs des

paramètres hôtes (e.g. la valeur de la charge CPU de *N1*) et réseaux (e.g. la valeur de la charge de la bande passante entre *N1* et *N2*) à jour sont calculées via NDS.

Cette section illustre l'intervention de deux modèles de coûts pour l'estimation du coût du plan d'exécution : le modèle de coûts qui réside sur un nœud nommé le modèle de coûts résident et le modèle de coûts embarqué dans un agent nommé le modèle de coûts embarqué [Hus05b]. Ces deux modèles de coûts permettent d'estimer des métriques d'agent. Ensuite, nous présentons la formule de coûts adaptée, en fonction des paramètres calculées par NDS, permettant d'estimer le coût d'exécution d'une jointure par hachage simple.

4.1. Modèle de coûts résident

Le modèle de coûts résident d'un nœud *N* contient le profil des relations connue par *N* et les formules de coûts.

4.2. Formule de coûts

Le temps de réponse d'une opération est une combinaison (i) du coût de traitement local et (ii) du coût de communication.

Le coût de traitement local (i) dépend du profil des opérandes, de l'indisponibilité des opérandes, des caractéristiques architecturales du nœud où l'opération est exécutée (e.g. temps CPU pour comparer un tuple) et de la charge de ce nœud (i.e. la charge CPU, la charge E/S et la charge mémoire). L'indisponibilité d'une relation est le temps d'attente pour recevoir l'opérande. On peut distinguer deux sortes d'indisponibilité [Ams96] : l'indisponibilité initiale INDi qui se produit pendant la production du premier tuple et l'indisponibilité continue INDc qui se produit pendant la production de chaque tuple. La charge d'un nœud dépend [Zhu00, Zhu03] : de la mémoire disponible, de la mémoire utilisée, du nombre d'E/S par seconde, du nombre de processus actifs et du nombre de processus suspendus...etc.

Le coût de communication (ii) dépend des caractéristiques de réseaux (i.e. la latence et la bande passante) et des quantités de données transférées inter-nœuds.

La formule estimant le coût d'exécution d'une jointure par hachage de deux relations de base : $T = R1 \infty R2$, exécutée par un agent mobile *AM*, est décrite dans la Figure 17 [Hus05a]. Cette formule est adaptée en fonction des valeurs des paramètres hôtes et réseaux calculés via NDS.

Dans cette formule *N1, N2, NH*, et *NS* désignent respectivement le nœud stockant *R1*, le nœud stockant *R2*, le nœud où un *AM* construit la table de hachage et le nœud où un *AM* sonde sa table de hachage. Les termes indicés par NDS (e.g. $SS0_{NDS}$) sont estimés en fonction des paramètres calculés par NDS (e.g. la charge CPU). Les autres termes utilisés dans la formule ont les significations suivantes :

- *coût_hachage* : le coût pour construire localement la table de hachage *TH* à partir de R_1,
- *coût_sondage* : le coût pour sonder localement *TH* avec *R2*. Les termes coût_hachage et coût_sondage peuvent être estimés en utilisant les formules décrites par [OZS 99],
- *coût_p (R)* : le coût nécessaire pour que *AM* reçoive l'opérande R,
- *coût_migration* : la somme du coût de migration de *AM* et du coût de transfert de la table de hachage. Le coût de migration d'un agent [Ism99] est la somme du coût de sérialisation (*coût_ser*), du coût de dé-sérialisation (*coût_deser*) et du coût de transfert d'un agent (*coût_trans*),
- $Init_{NDS}(N1, N1)$: le temps pour établir une communication entre deux nœuds *N1* et *N2* estimé en fonction de la valeur de la latence entre *N1* et *N2*, calculée par NDS,
- $Trans_{NDS}(N1, N2)$: le temps pour envoyer une page de *N1* vers *N2* estimé en fonction de la valeur de la bande passante entre *N1* et *N2*, calculée par NDS,
- $coût\_trans_{NDS}$: Le coût de transfert de la relation *T* dépend principalement de la localisation de l'agent consommant *T* et des caractéristiques réseaux, calculés par NDS,
- $SS0_{NDS}$: coût initial pour la lecture séquentielle estimé en fonction de la charge du disque, calculée par NDS,
- $SS1_{NDS}$: coût unitaire pour retrouver un tuple estimé en fonction de la charge CPU d'un nœud, calculée par NDS,
- $SS2_{NDS}$: coût pour traiter un tuple résultat estimé en fonction de la charge de mémoire d'un nœud, calculée par NDS,
- *sel (P)* : désigne le facteur de sélectivité. Celui-ci est déterminé en fonction du prédicat de la jointure et il peut être calculé par des formules décrites dans [Ozs99].

4.3. Modèle de coûts embarqué

Le modèle de coûts embarqué [Hus05b] dans un agent est fourni à l'agent pendant la phase d'optimisation. Le modèle de coûts d'un agent est composé de trois unités : (i) l'espace de migration (i.e. *N1* et *N2*), (ii) des métadonnées (estampillées par une date) décrivant les relations de l'opérateur (exécuté par un agent) et (iii) des métadonnées statiques [Sam07] décrivant les nœuds de l'espace de migration. Un agent n'a pas besoin d'embarquer les formules de coûts estimant ses métriques parce qu'elles sont répliquées sur tous les nœuds. L'espace de migration (i) est l'ensemble de nœuds où l'agent peut migrer. Les métadonnées décrivant les relations (ii) est important à embarquer car le nœud où l'agent construit sa table de hachage ne possède pas forcément le profil de la deuxième relation R2. Pour la même raison, les métadonnées statiques (iii) décrivant *N1* et *N2* sont importantes à embarquer (surtout *N2*) car le nœud *N1* (où se trouve un agent) ne dispose pas forcément des métadonnées statiques décrivant *N2*. Quant aux valeurs de paramètres hôtes et réseaux à jour, ils sont capturés par un agent grâce à NDS.

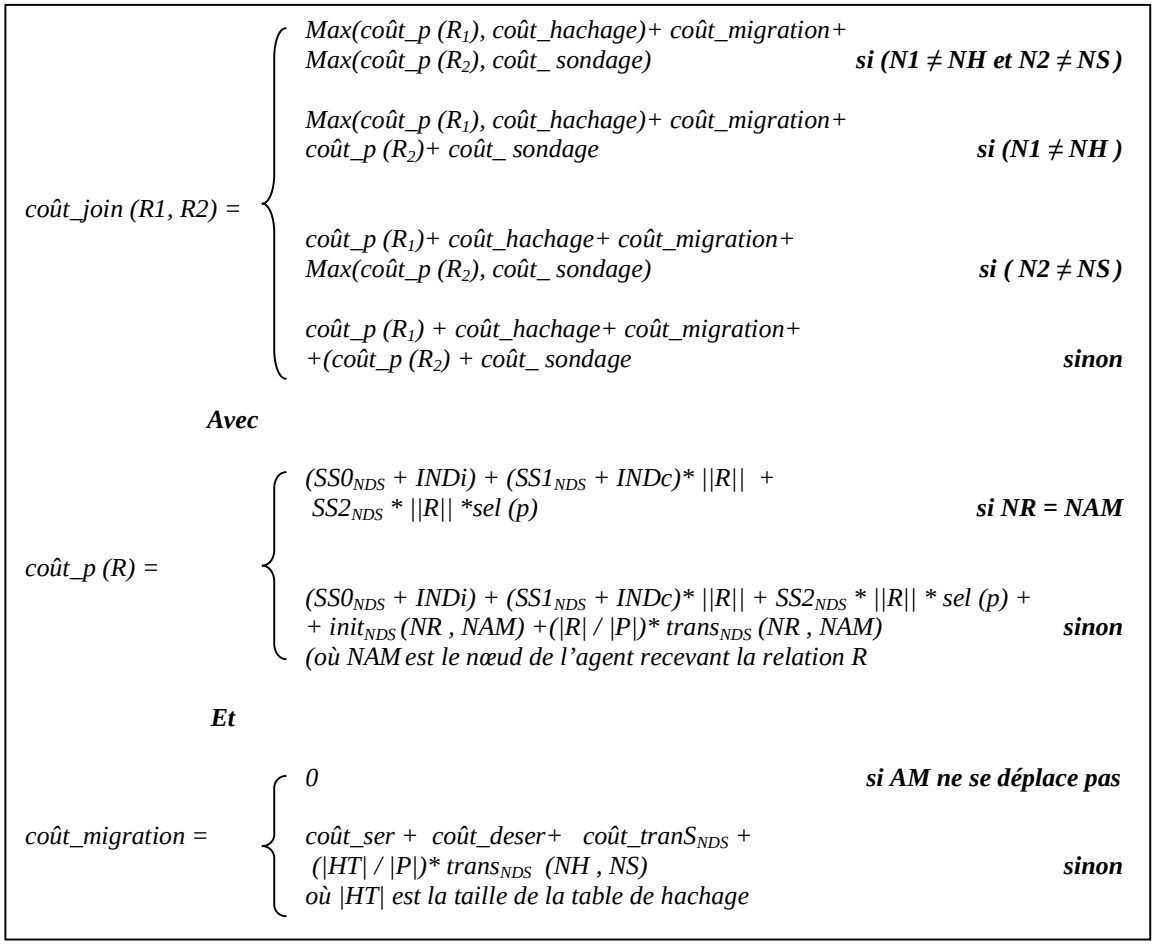

Figure 17 : Formule de coûts pour une jointure mobile par hachage [Hus05a] adaptée en fonction des valeurs des paramètres hôtes et réseaux calculés via NDS

5. Intégration des informations de monitoring dans une jointure mobile

5.1. Détails d'implémentation de NDS avec les agents mobiles

Pour pouvoir calculer un ensemble de valeur de paramètres (e.g. valeur de la charge CPU d'un nœud, valeur de bande passante entre deux nœuds), un agent mobile doit d'abord préciser l'ensemble des hôtes (e.g. *N1*, *N2*) dont il s'intéresse. Un agent mobile peut utiliser NDS via une Application Programming Interface (API). Les services de monitoring et les paramètres disponibles sont déclarés dans un fichier de configuration de type Java Name and Directory Interface (JNDI). Ainsi, le NDS est configuré suivant l'ensemble de paramètres disponibles. Il permet de calculer tous les paramètres demandés suivant un format unique et renvoie la réponse à

son client (ici c'est un agent mobile). Un paramètre est décrit par : son nom, le type de la mesure (e.g. un réel, un double) et la commande pour calculer sa valeur. Par exemple, la déclaration du paramètre de la charge CPU d'un nœud (rep. la bande passante entre deux nœuds) observée par NWS dans le fichier de configuration NDS est montrée dans la Figure 19 (resp. la Figure 18).

Quand nous avons utilisé NDS, ce dernier intégrait uniquement le NWS. NWS permet d'observer des paramètres réseaux et hôtes. Les paramètres observés par NWS (et fourni par NDS) sont la charge CPU d'un nœud, le débit d'E/S d'un nœud, la charge mémoire d'un nœud, et la latence et la bande passante entre deux nœuds. L'intégration d'autres outils de monitoring (e.g. MDS) est tout à fait faisable. Actuellement, les deux outils NWS et MDS sont intégrés dans NDS. Pour plus d'information sur NDS, ce dernier est disponible en ligne [Gos07b].

```
<parameter>
<name>
  metrics
</name>
<value>
  NWSCpuLoad
  The CPU load value of N
  Parameter : %N = IP or name of target
  nws_extract CpuLoad -n1 -h0 –f mea %N
  double
  80.341
</value>
</parameter>
```

Figure 18 : Déclaration du paramètre de la charge CPU de NWS dans le fichier de configuration NDS

```
<parameter>
<name>
  metrics
</name>
<value>
  NWSBandwidth
  The bandwidth value between N1 and N2
  Parameters : %N1, %N2 = IP or name of targets
  nws_extract bandwidth -n1 -h0 –f mea %N1 %N2
  double
  80.341
</value>
</parameter>
```

Figure 19 : Déclaration du paramètre de la bande passante de NWS dans le fichier de configuration NDS

5.2. Interactions d'un agent mobile avec le nœud local et NDS

Dans cette sous-section, nous souhaitons présenter les interactions d'un agent mobile avec le nœud local (i.e. où il construit la table de hachage de *R1*) et NDS. Lors de son installation, un agent mobile *AM* commence d'abord par construire la table de hachage *TH* de *R1* sur *N1*. Il peut ainsi savoir le profil précis à la fin de la construction de *TH* (pour le moment il n'interagit pas ni avec son *EM*, ni avec NDS). Lors de l'étape de décision, *AM* fournit au modèle de coûts (i) son espace de migration (ii) le profil précis de *R1* et le profil de *R2*, (iii) les caractéristiques statiques de *N1* et *N2*, (iv) les valeurs de certains paramètres hôtes de *N1* et *N2* et les valeurs de certains paramètres réseaux reliant *N1* à *N2*. (i), (ii) et (iii) constituent le modèle de coûts embarqué dans *AM*. (iv) est calculé grâce à NDS.

Le profil précis de *R1* et le profil de *R2* (i) sont important pour pouvoir calculer le profil de *T*. Ces profils sont estampillés par une date. Si le nœud courant stocke dans son catalogue local le profil de *R2*, l'agent choisit le profil le plus récent. Ensuite, *AM* demande au modèle de coûts de calculer le coût d'exécution d'une partie de la jointure mobile en fonction du coût de l'exécution *CE* sur *N1* (nommé CE_{N1}) ou sur *N2* (nommé CE_{N2}). Si le coût de traitement de l'étape de sondage sur *N1* plus le coût de l'envoi de *R2* (de *N2* à *N1*), nommé CE_{N1}, est inférieur au coût de l'envoi de la table de hachage de *R1* (de *N1* à *N2*) plus le coût de traitement de l'étape de sondage sur *N2* ainsi que le coût de l'envoi du résultat *T* (de *N2* à *N1*), nommé CE_{N2}, alors l'agent décide d'exécuter *J* sur *N1*. Sinon, il décide d'exécuter *J* sur *N2*. Les coûts CE_{N1} et le CE_{N2} sont calculés grâce aux formules de coûts présentes sur le nœud local.

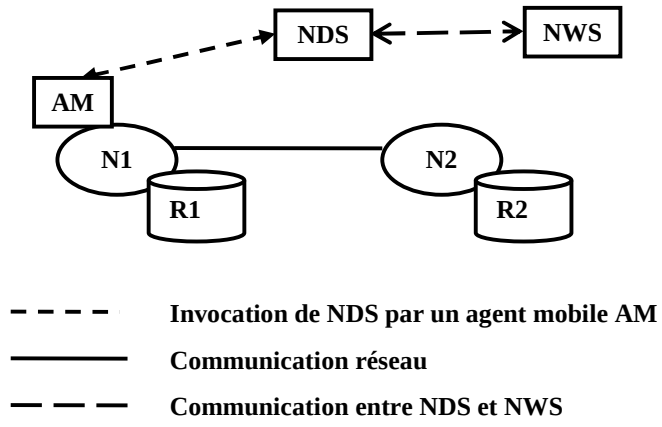

Figure 20 : Scénario d'utilisation de NDS par un agent mobile AM (juste avant la prise de décision)

Afin de mieux illustrer l'exploitation des paramètres hôtes et réseaux (calculés via NDS) et utilisé par un agent mobile *AM*, nous présentons par des diagrammes de séquence UML le comportement de la méthode proposée dans les deux cas :

(i) Exécution de la phase du sondage sur N1,
(ii) Exécution de la phase du sondage sur N2.

Dans le cas (i), l'agent décide d'exécuter la jointure mobile *J* sur *N1*. Ainsi, il contacte *N2* via son interface en lui demandant d'envoyer *R2* à *N1*. Ensuite, l'étape de sondage peut démarrer dès la réception du premier tuple de *R2*. Enfin, la matérialisation peut être effectuée sur *N1* au fur et à mesure de la production des tuples résultats de la relation *T* issus de l'étape de sondage.

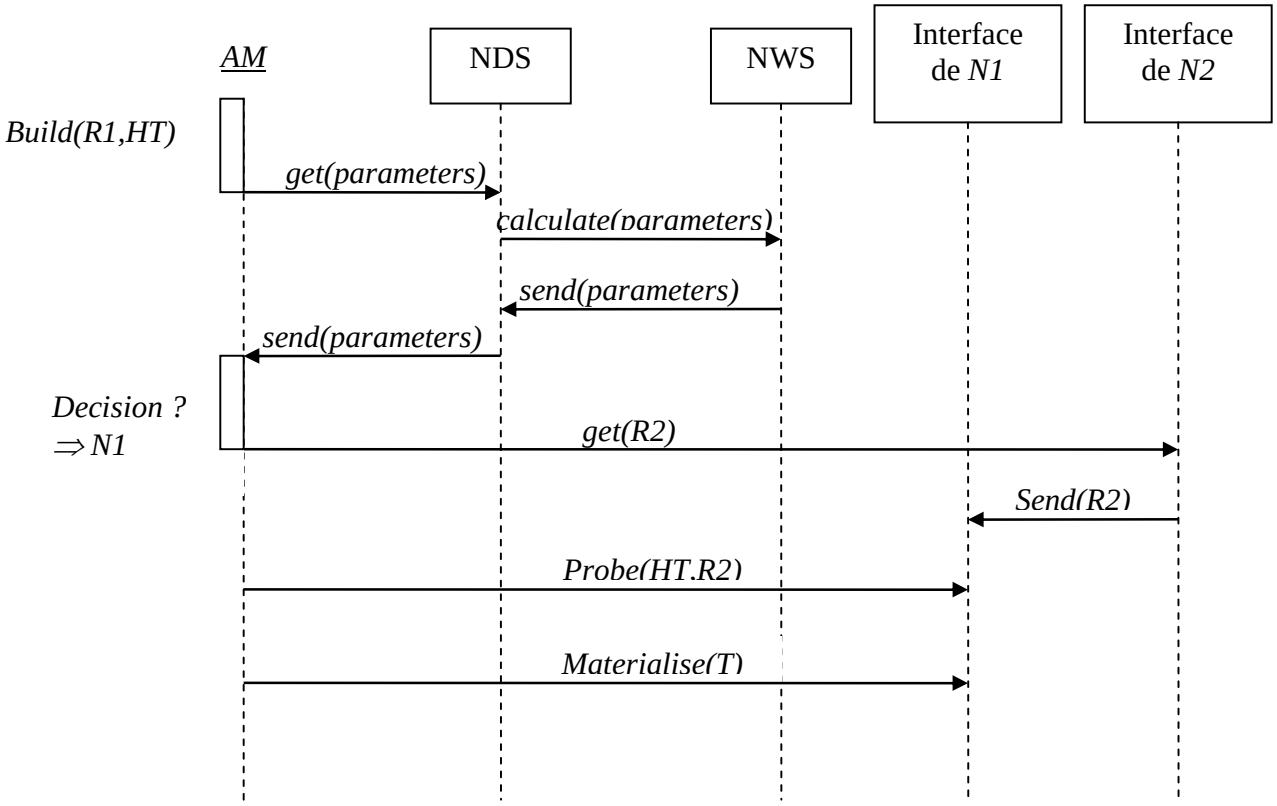

Figure 21 Illustration de l'utilisation de NDS par un agent mobile AM lors de l'exécution de la phase de sondage sur *N1*

Dans le cas (ii), l'agent décide d'exécuter la jointure mobile *J* sur *N2*. Ainsi, il contacte *N1* via son interface en lui demandant d'envoyer la table de hachage *HT* de *R1* à *N2*. Ensuite, l'étape de sondage peut démarrer une fois la table de hachage est complètement envoyé à *N2*. Le résultat est ensuite envoyé au fur et à mesure de sa production et il est matérialisé sur *N1*.

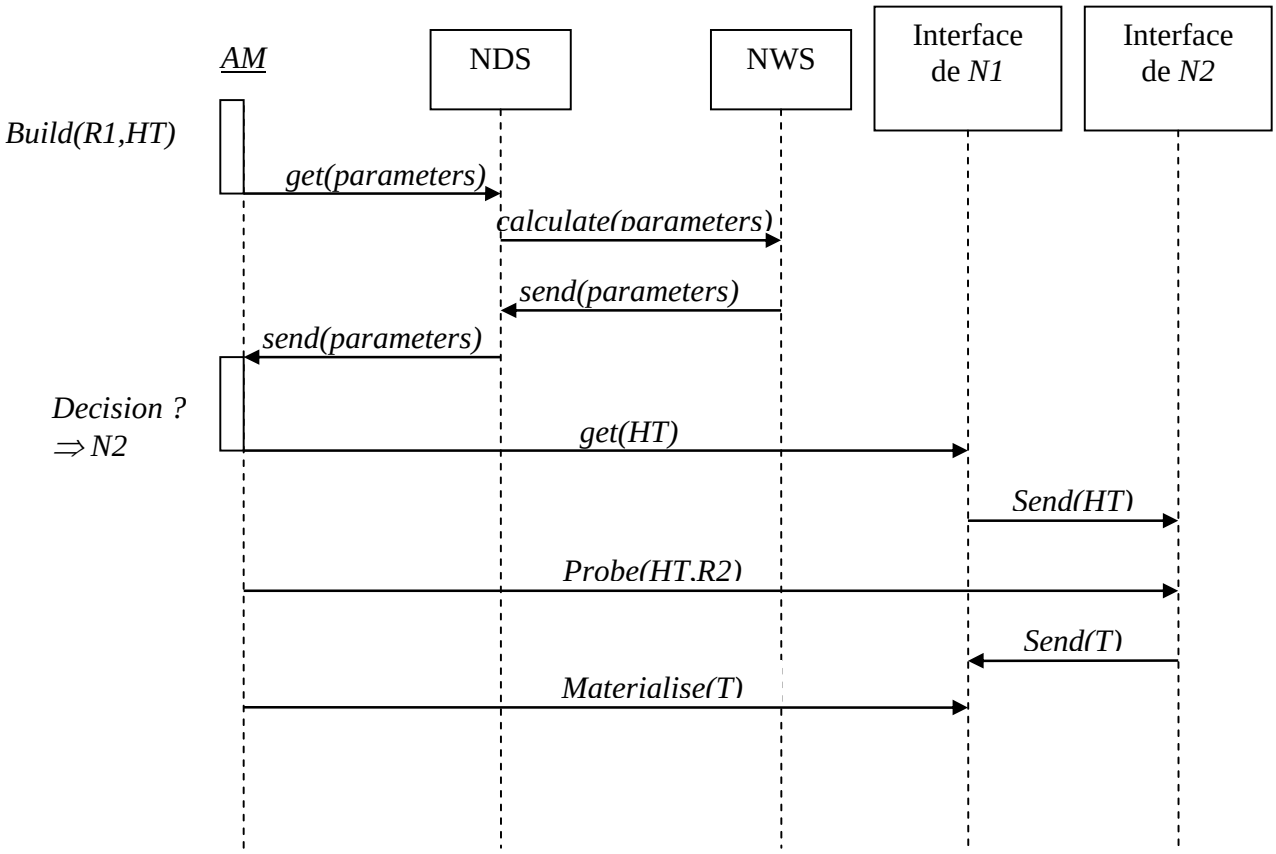

Figure 22 Illustration de l'utilisation de NDS par un agent mobile AM lors de l'exécution de la phase de sondage sur *N2*

6. Conclusion

Dans ce chapitre, nous avons proposé une méthode permettant la prise en compte de la variation de valeurs des paramètres hôtes et réseaux dans le calcul du temps de réponse de la jointure mobile. Dans un premier temps, nous avons décrit le problème adressé dans ce chapitre. Plus précisément, il s'agit de permettre aux agents mobiles une prise de décisions efficace en présence de la variation des valeurs des paramètres hôtes et réseaux. Nous avons précisé qu'un agent se base sur les informations qui figurent dans le modèle de coûts. En effet, ces informations sont souvent calibrées dans les approches classiques (e.g. modèle de coûts calibré) et peuvent être obsolètes. Ainsi, nous proposons d'utiliser des informations de monitoring permettant de capturer les valeurs des paramètres hôtes et réseaux à jour. Ensuite, nous avons présenté puis analysé les limites de l'utilisation des services de monitoring (dans notre contexte) employés aujourd'hui. En effet, la plupart des solutions actuelles consistent à définir un seul service pour chaque type de paramètre. Utiliser un outil de monitoring pour chaque paramètre est assez lassant dans un système à grande échelle et en particulier pour des agents mobiles. Ceci implique qu'un agent doit programmer toutes les méthodes d'invocation des outils de monitoring dont il a besoin. Une solution pour éviter le problème ci-dessus consiste à utiliser un seul service (invocable via une seule interface) pour tous les outils de monitoring comme le Network Distant Service (NDS). NDS permet de calculer la meilleure valeur estimée d'un paramètre hôte ou réseau. Puis, nous avons présenté le modèle de coûts et nous avons détaillé la formule de coûts adaptée, en fonction des paramètres calculés par NDS, permettant d'estimer le coût d'exécution d'une jointure par hachage simple. Enfin, nous avons montré comment l'intégration des informations de monitoring peut être effectuée dans une jointure mobile.

Le prochain chapitre est consacré à l'évaluation des performances de nos propositions.

Chapitre V. Évaluation des performances

1. Introduction

L'objectif de ce chapitre est d'évaluer les performances de nos propositions. Nous évaluons, tout d'abord, la méthode proposée [Sam09] de découverte de ressources (de type sources de données) par rapport aux méthodes actuelles de découverte de ressources. Ensuite, nous évaluons les performances de l'opérateur de jointure mobile par hachage en l'absence puis en la présence d'un service de monitoring. Enfin, nous analysons et nous discutons des résultats de ces évaluations.

2. Évaluation des performances de la méthode de découverte de ressources

Dans cette section, nous nous intéressons à l'évaluation des performances de notre méthode de découverte de ressources utilisant Plusieurs THDs (la méthode PTHD) par rapport à : (i) la méthode de découverte de ressources, basée sur une techniques Super-Pair [Mas05] (la méthode SP) et (ii) la méthode de découverte de ressources, basée sur l'utilisation d'une Seule THD pour tout le système [Dov03, Gal03] (la méthode STHD). Les ressources considérées dans les trois méthodes sont des relations. Chaque relation est décrite par un ensemble de métadonnées.

Nous évaluons notre méthode PTHD uniquement par rapport à la méthode SP et la méthode STHD car une comparaison (par simulation) récente existe déjà dans [Mas08] entre la méthode SP, les méthodes P2P non-structurées et les méthodes hiérarchiques. Cette comparaison a montré dans plusieurs scénarios la supériorité de la méthode SP.

2.1. Choix des paramètres pour la comparaison des méthodes de découverte

La comparaison est effectuée suivant les deux paramètres : le nombre de messages de découverte par seconde (P1) et le nombre de nœuds qui se connectent ou qui se déconnectent du

système (P2). Le paramètre (P1) détermine si une méthode peut passer à l'échelle en présence d'un grand nombre de messages de découverte. Le deuxième paramètre (P2) détermine l'impact d'un effet churn sur une méthode de découverte.

Les autres paramètres (ou critères) présentés dans le tableau de comparaison global (du chapitre II) ne seront pas étudiés. Plus précisément il s'agit de la découverte fiable, du mécanisme de la propagation de la découverte et de la capacité de traiter des requêtes complexes. La découverte est fiable pour les 3 méthodes SP, STHD et PTHD (voir le tableau de comparaison global). Aussi, le mécanisme de propagation figure dans le tableau de comparaison global pour ces 3 méthodes. Enfin, le traitement des requêtes complexes (spécifiques aux ressources de calcul) est hors notre contribution.

2.2. Objectifs

L'objectif de l'évaluation des performances de la méthode PTHD est d'étudier l'impact de P1 sur le temps moyen de réponse [Mas08] pour la découverte de relations avec la méthode PTHD et la méthode SP (resp. la méthode STHD). Ce temps est le temps moyen que met le système pour traiter un message de découverte de relations. Ainsi, nous souhaitons étudier l'intervalle où la méthode PTHD est meilleure que la méthode SP (resp. meilleure que la méthode STHD) en fonction du nombre de messages de découverte par seconde.

Ensuite, nous souhaitons étudier l'impact de P2 sur le nombre de messages générés avec la méthode PTHD et STHD. Nous n'étudions pas dans cette partie l'impact de P2 sur la méthode SP car si un super-pair tombe en panne, l'OV devient inutilisable et inaccessible par le système.

2.3. Modèle de simulation

L'étude du comportement des trois méthodes (PTHD, SP, STHD) doit être effectuée sur un grand nombre de nœuds (qui peuvent atteindre une dizaine de milliers). Ne possédant pas un tel matériel, l'évaluation est effectuée par simulation. Le simulateur permet de publier périodiquement de métadonnées décrivant des relations et de les découvrir. Chaque simulation est effectuée une dizaine de fois puis nous calculons le temps moyen de réponse. Les nœuds peuvent joindre ou quitter le système à tout instant. Afin de rendre la comparaison équitable, nous supposons la même distribution de relations [Mas08] dans toutes les simulations (i.e. on a le même nombre de relations dans chaque OV et chaque nœud stocke en moyenne le même nombre de relations).

Dans la première partie de simulations (étude de P1), nous calculons le temps moyen de réponse. Le temps de réponse inclut le coût de communication réseau et le coût du traitement local pour les trois méthodes (SP, STHD, PTHD). Le coût de communication réseau dans la méthode SP vaut le coût d'un saut local si la découverte intra-OV réussit. Sinon elle vaut la somme du coût du saut local et des sauts globaux entre deux nœuds super-pairs. Le coût de communication réseau pour une méthode basée sur l'utilisation des THDs est de complexité $O(Log(N))$ [Sto01] avec N le nombre de nœuds dans le système. Le coût du traitement local est le coût nécessaire pour la comparaison des métadonnées décrivant les relations référencées dans une requête de découverte, avec les métadonnées décrivant les relations d'un nœud local contacté durant le processus de la découverte [Mas08].

Dans la deuxième partie (étude du P2), nous calculons le nombre de messages générés lors de la connexion ou de la déconnexion des nœuds. Nous reprenons les paramètres du système P2P structuré Chord [Sto01] vu que ce dernier a été largement utilisé et considéré comme le système P2P référence. Dans les THDs du système Chord, la connexion ou la déconnexion d'un nœud génère $O(Log^2(N))$ messages avec N le nombre de nœuds dans le système [Sto01]. Dans la méthode PTHD, la connexion d'un nœud génère également $O(Log^2(M))$ messages (avec M le nombre des nœuds dans une OV) plus le nombre de messages envoyés pour l'établissement des points de sortie aux OVs distantes. La déconnexion d'un nœud vaut aussi $O(Log^2(M))$ messages avec la méthode PTHD car nous avons décidé d'utiliser une stratégie de mise à jour paresseuse du système d'adressage. En effet, lorsqu'un nœud quitte son OV, ce nœud n'indique pas sa déconnexion aux autres OVs dans le système car ceci peut être complètement inutile et peut générer une inondation dans les OVs distantes. Le système d'adressage permet l'interconnexion entre toutes les OVs et il est mis à jour lors de la découverte inter-OV de relations et lors de l'arrivée de nouveaux nœuds (pour plus de précision voir le chapitre III).

Pour les trois méthodes (SP, STHD, PTHD), le coût d'un saut est entre n'importe quel deux nœuds est estimé à 30 ms [Sto01] et le coût du traitement local d'une requête est estimé à 0,2 ms [Has05]. Nous supposons que le système est constitué de 10 OVs. Chaque OV contient en moyen 1000 nœuds physiques.

2.4. Impact du nombre de messages de découverte par seconde sur le temps moyen de réponse

Dans cette partie, nous faisons varier le nombre de messages de découverte par seconde. Ces messages sont lancés par des nœuds choisis au hasard suivant deux scénarios : le premier suppose que les relations à découvrir appartiennent à l'OV locale (i.e. découverte intra-OV) puis le deuxième suppose que les relations à découvrir appartiennent à une autre OV (i.e. découverte inter-OV). Ces scénarios sont valables pour la méthode SP et la méthode PTHD.

Dans la méthode STHD, on ne distingue pas la différence entre une découverte intra-OV ou inter-OV car la THD est construite pour tout le système.

2.4.1. La méthode SP VS la méthode PTHD

La Figure 23 présente le temps moyen de réponse pour une découverte intra-OV (resp. inter-OV) en fonction du nombre de messages de découverte par seconde. Nous avons observé des valeurs différentes :

- Cas SP : lorsque le nombre de messages de découverte augmente, le temps moyen de réponse augmente d'une manière significative.
- Cas PTHD : lorsque le nombre de messages de découverte augmente, le temps moyen de réponse est presque constant.

Dans le cas SP, les messages de découverte sont traités d'une manière séquentielle par un super-pair ce qui augmente le temps moyen de réponse (i.e. le coût du traitement local et réseau). Dans le cas PTHD, les messages de découverte sont traités en parallèle dans la méthode PTHD permettant ainsi d'avoir un temps moyen de réponse presque constant.

La méthode SP est meilleure que la méthode PTHD pour un nombre de messages de découverte par seconde inférieur à 20 messages par seconde lors d'une découverte intra-OV (resp. 21 messages par seconde lors d'une découverte inter-OV).

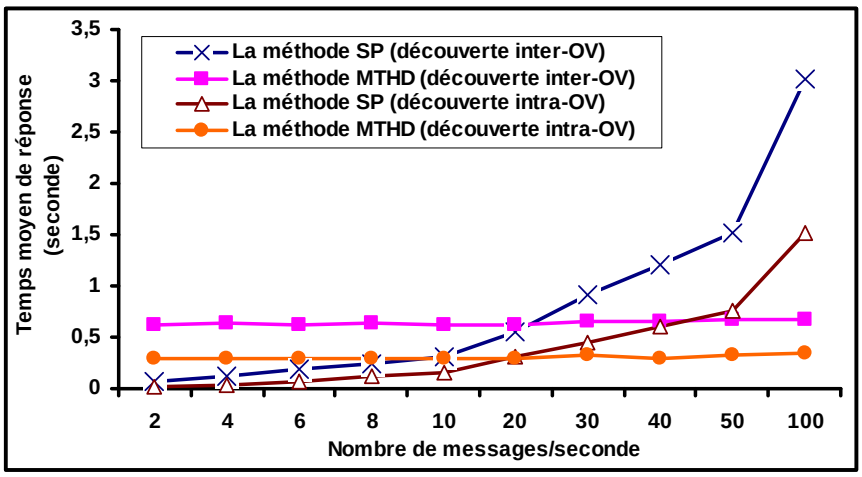

Figure 23 : Impact du nombre de messages de découverte par seconde sur le temps moyen de réponse

Le facteur d'accélération est d'environ 5 pour 100 messages par seconde lors d'une découverte intra-OV ou inter-OV (le FA est le rapport entre le temps moyen de réponse avec la méthode SP sur le temps moyen de réponse avec la méthode PTHD). Ce FA est calculé pour un débit de 0,1 message par seconde et par nœud i.e. un message toutes les 10 secondes, dans une OV contenant en moyen 1000 nœuds, ce qui est relativement petit pour un système à grande échelle. Par conséquence, la méthode SP ne peut pas passer à l'échelle des OVs constituées d'un nombre important de nœuds surtout que le temps moyen de réponse peut devenir très grand pour un débit relativement petit de messages de découverte dans une OV.

2.4.2. La méthode STHD VS la méthode PTHD

La Figure 24 présente le temps moyen de réponse en fonction du nombre de messages de découverte par seconde pour la méthode STHD et la méthode PTHD.

Dans le cas d'une découverte intra-OV, la méthode PTHD est meilleure que la méthode STHD d'environ 13%. En effet, le nombre de nœuds dans une OV est plus petit que le nombre de nœuds dans tout le système. Ainsi, le temps moyen de réponse lors d'une découverte intra-OV avec la méthode PTHD est toujours plus petit que le temps moyen de réponse lors de l'utilisation de la méthode STHD.

Dans le cas d'une découverte inter-OV, la méthode STHD est meilleure que la méthode PTHD d'environ 17%. Dans la méthode PTHD, si la découverte dans la THD locale échoue alors la découverte est lancée dans toutes les autres THDs (il est indispensable de découvrir toutes les

relations afin de pouvoir exécuter une requête de type SQL ou autre). Par conséquent, le temps moyen de réponse avec la méthode PTHD est toujours plus grand que le temps moyen de réponse avec la méthode STHD pour une découverte inter-OV.

Enfin, nous constatons que pour les trois courbes le coût moyen de la découverte d'une relation est presque constant, sauf pour une découverte inter-OV avec la méthode PTHD. En effet, le coût moyen n'est pas constant dans ce cas car la découverte inter-OV nécessite de contacter des points de sortie qui peuvent être éventuellement déconnectés ce qui peut provoquer un léger surcoût.

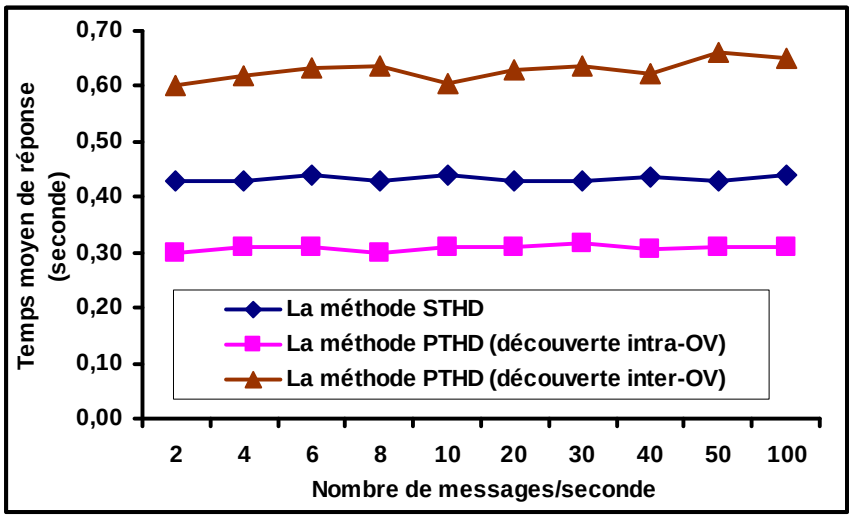

Figure 24 : Impact du nombre de messages de découverte par seconde sur le temps moyen de réponse

2.5. Impact du nombre de nœuds qui se connectent ou qui se déconnectent du système sur le nombre de messages échangés dans le système

La Figure 25 présente le nombre de messages générés en fonction du nombre de nœuds qui se connectent ou qui se déconnectent avec la méthode PTHD et la méthode STHD.

La méthode PTHD est toujours meilleure que la méthode STHD car le nombre de nœuds dans une OV est inférieur au nombre de nœuds dans le système. Dans le cas de la connexion ou de la déconnexion d'un nœud (resp. de 20 nœuds), la méthode STHD génère à peu près 32,4% (resp. 50,7%) des messages en plus que la méthode PTHD. Dans la méthode PTHD, le surcoût de la connexion par rapport à la déconnexion d'un nœud, est lié au fait que la connexion d'un nouveau nœud nécessite un nombre supplémentaire de messages pour l'établissement des points de sortie

de ce nouveau nœud à toutes les autres OVs distantes dans le système. Par contre, lors de la déconnexion d'un nœud, ce dernier n'envoie pas aux OVs distantes l'information sur sa déconnexion car le système d'adressage emploie une stratégie de mise à jour paresseuse.

Le facteur d'accélération (calculé en fonction du nombre de messages échangés dans le système) de la méthode PTHD par rapport à la méthode STHD, est compris entre 1.77 (pour 1 nœud connecté ou déconnecté) et 2.04 (pour 20 nœuds connectés ou déconnectés).

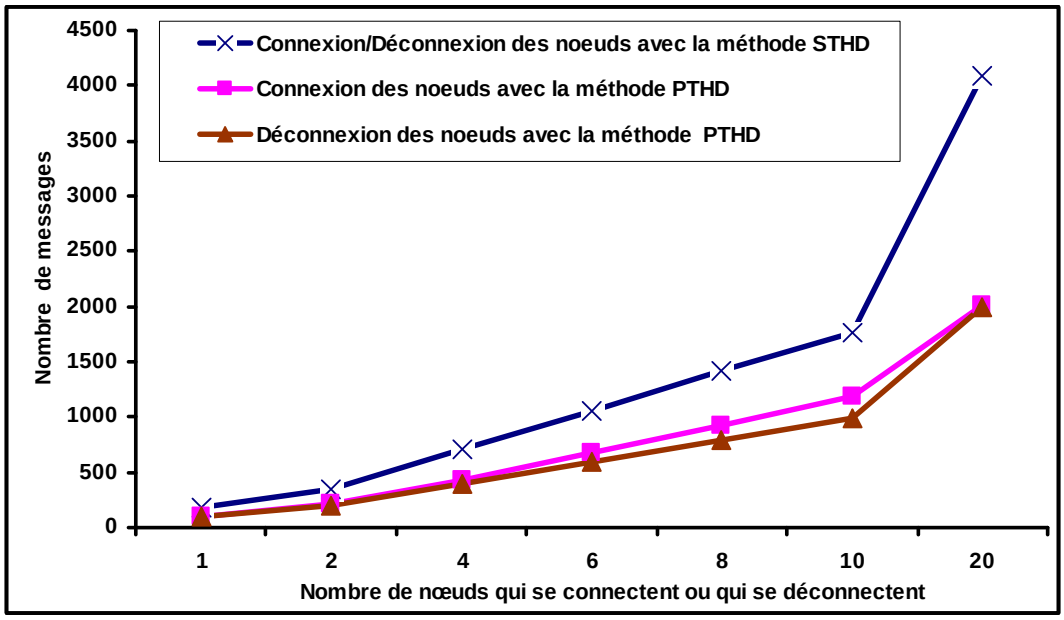

Figure 25 : Impact du nombre de nœuds qui se connectent ou qui se déconnectent sur le nombre de messages échangés dans le système

3. Évaluation des performances de l'opérateur de jointure mobile par hachage en présence d'un service de monitoring

3.1. Objectifs et hypothèses

Dans cette section, nous nous intéressons à l'évaluation des performances de l'opérateur de jointure mobile par hachage en présence d'un service de monitoring, le Network Distant Service (NDS). Nous rappelons qu'un opérateur mobile peut migrer d'un nœud à un autre nœud pour réagir aux erreurs d'estimation et à la variation des valeurs des paramètres hôtes et réseaux. Toutes les mesures sont effectuées en fixant des erreurs d'estimation sur une des deux relations d'une jointure, il s'agit de la première relation *R1*, car il est possible de réviser le profil de *R1* durant la construction de sa table de hachage. On considère que les relations ne sont pas

répliquées, chaque opérande de la jointure réside sur un seul nœud du système. Nous souhaitons comparer :

(i) Le temps d'exécution d'une jointure mobile en absence du NDS (Modèle de coûts calibré).
(ii) Le temps d'exécution d'une jointure mobile en présence du NDS (calcul des valeurs des paramètres hôtes et réseaux durant l'exécution).

Dans le premier cas (i), le modèle de coûts calibré utilise des paramètres hôtes « CPU » et réseaux « bande passante » qui sont déjà calculés lors d'une étape de calibration. Dans le deuxième cas (ii), les valeurs de ces paramètres sont calculées pendant l'exécution d'une jointure mobile plus précisément pendant la phase de décision.

Dans les deux cas (i) et (ii), nous supposons que la latence est négligée car la taille du premier paquet de données (i.e. le premier tuple) envoyé est très petite devant la taille totale de données envoyées. Les paramètres E/S seront calibrés dans les deux cas (i) et (ii).

Dans le reste de cette section, nous décrivons l'environnement et le prototype d'expérimentation, les résultats obtenus en calibration et enfin les résultats de nos expériences en présence du NDS.

3.2. Environnement et prototype d'expérimentation

Considérons deux relations *R1* et *R2* résidantes respectivement sur les nœuds *N1* et *N2* avec un nombre de tuples estimés à 10000 et 20000. Considérons également une jointure par hachage simple *J* : $T = R1 \infty R2$. Le facteur de sélectivité de *J* est estimé à 1.5/max (||*R1*||, ||*R2*||) (où ||*Ri*|| est la cardinalité de la relation *Ri*, qui correspond au nombre de tuples). Le nombre de tuples moyen par page est estimé à 32 tuples par page [She93]. On suppose que *T* doit être matérialisé sur *N1*.

Comme nous avons déjà indiqué dans le chapitre IV, un agent peut prendre une décision sur le choix du nœud de migration. La décision de migration d'un agent est prise en fonction de la comparaison du coût de traitement de l'étape de sondage sur *N1* plus le coût d'envoi de *R2* sur *N1* avec, le coût d'envoi de la table de hachage de *R1* sur *N2* plus le coût du traitement de l'étape de sondage sur *N2* ainsi que l'envoi du résultat sur *N1*. Nous supposons que ces différents coûts dépendent principalement des valeurs de la charge CPU de *N1* et *N2* et des valeurs de la bande passante entre *N1* et *N2*. Nous ne traitons pas les autres paramètres (e.g. E/S). Ainsi, nous nous

intéressons à l'étude de l'impact de la CPU et de la bande passante sur le comportement de la jointure mobile.

Nous avons mené les mêmes expériences dans deux environnements d'expérimentation pour observer le comportement de *J* en fonction de la variation des valeurs de la charge CPU et de la bande passante. Il s'agit des deux environnements :

(i) **Environnement local** : il est constitué de deux stations de travail *N1* (Linux Redhat, Fréquence du processeur = 2.8 Ghz, cache = 1024 KO, mémoire = 512 MO) et *N2* (Linux Fedora Core, Fréquence du processeur = 2.8 Ghz, cache = 256 KO, mémoire = 512 MO) interconnectés à un réseau local avec une bande passante de 10 MO/seconde.

(ii) **Environnement à grande échelle** : il est constitué de deux stations de travail *N1* (Linux Redhat, Fréquence du processeur = 2.8 Ghz, cache = 1024 KO, mémoire = 512 MO) et *N2* (Linux Redhat, Fréquence du processeur = 512 Mhz, cache = 512 KO, mémoire = 128 MO) interconnectés via Internet. La première se situe à l'IRIT (Toulouse), la deuxième se situe au Liris (Lyon) avec une bande passante de 450 KO/seconde.

La bande passante entre Toulouse et Lyon peut apparaître un peu grande par rapport aux valeurs de la bande passante dans un environnement à grande échelle où la valeur de la bande passante peut atteindre quelques KO par seconde. La simulation d'un environnement à grande échelle est tout à fait réalisable mais nous préférons utiliser une réelle plateforme d'expérimentation interconnectant plusieurs nœuds entre Toulouse et Lyon dans le cadre du projet GGM (Grid for Geno Medecine) [Pie07]. Si on fait le rapport entre la valeur de la bande passante en environnement local (10 MO/s) et en environnement à grande échelle (450 KO/s), nous constatons que cette valeur a diminué d'environ 22 fois ce qui permet d'étudier le comportement de la jointure mobile dans les deux cas ci-dessus.

Afin de mener nos expériences, nous avons mis en œuvre une plate-forme d'expérimentation où chaque opérateur mobile s'exécute sur une Machine Virtuelle Java (JDK 1.4.2).

3.3. Résultats obtenus par calibration en fonction des erreurs d'estimation [Hus05a]

En considérant la jointure mobile, l'évaluation des performances dans [Hus05a] a prouvé -pour le prototype présenté dans la section précédente- que l'exécution est optimale sur *N1* pour des erreurs d'estimation sur ||*R1*|| de 0% et -30%. Pour des erreurs d'estimation sur ||*R1*|| de -

60% et -90%, l'exécution est optimale sur *N2*. Ce résultat est dû à la taille des relations *R1*, *R2* et *T*. L'interprétation est la suivante :

- Pour une erreur d'estimation sur ||*R1*||comprise entre l'intervalle [-30%; 0%] l'exécution est optimale sur *N1* à cause de l'inégalité : $|R1| + |T| > |R2|$ **(A)**
- Pour une erreur d'estimation sur ||*R1*||comprise entre l'intervalle [-90% ; -30%[l'exécution est optimale sur *N2* à cause de l'inégalité : $|R1| + |T| < |R2|$ **(B)**
  (Où |*Ri*| est la taille de la relation Ri, qui correspond au nombre de pages)

Les travaux de [Hus05a] considèrent uniquement des erreurs d'estimation sur ||*R*1|| mais ne tiennent pas compte de la variation des valeurs des paramètres de la charge CPU (V1) et de la charge de la bande passante (V2). Dans ce qui suit, nous présentons nos expériences menées en environnement local puis en environnement à grande échelle.

3.4. Expériences en environnement local

Dans cette section, nous décrivons les résultats des expériences menées en environnement local. Dans ces expériences, nous allons considérer (i) les erreurs d'estimation sur ||R1|| et, (ii) la variation des valeurs de la charge CPU (V1) puis la charge de la bande passante (V2). Nous comparons le temps de réponse de *J* en absence et en présence du NDS.

3.4.1. Impact de la variation des valeurs de la charge CPU sur le temps de réponse

Dans cette partie, nous souhaitons étudier l'impact de V1 sur le temps de réponse de *J*. NDS sera utilisé uniquement pour le calcul des valeurs de la charge CPU. Le paramètre de la bande passante sera calibré afin de permettre une étude en fonction du paramètre souhaité et afin d'isoler chaque paramètre.

Exécution avec des erreurs d'estimation de 0% et -30% sur ||R1||

Pour des erreurs d'estimation sur ||*R1*|| comprises entre 0% et -30%, le plan d'exécution optimal de *J* est de construire la table de hachage sur *N1*, puis de recevoir *R2* sur *N1* afin d'effectuer l'étape du sondage (inégalité (A)). Ainsi, si la charge CPU de *N2* varie, le plan d'exécution optimal de *J* reste le même. Par contre, lorsque la valeur de la charge CPU de *N1* varie, il est possible de déplacer l'étape du sondage de *J* de *N1* à *N2* afin d'avoir un temps de

réponse proche de l'optimale. Pour cela, nous étudions le comportement des agents en fonction de la variation des valeurs de la charge CPU de *N1*.

La Figure 26 présente le temps de réponse de *J* en l'absence et en la présence du NDS avec 0% d'erreur d'estimation sur ||*R1*|| (resp. avec -30% d'erreur d'estimation sur ||*R1*||) et en fonction de la variation des valeurs de la charge CPU de *N1* par rapport à *N2* (noté *N1*/*N2*) de 0% à 400% (i.e. *N1* est 4 fois plus chargé que *N2*). Nous avons observé deux comportements différents en l'absence et en la présence du NDS.

(i) Exécution en absence du NDS : Dans ces expériences, quelque soit la variation des valeurs de la charge CPU, l'étape du sondage est effectuée sur *N1*.

(ii) Exécution en présence du NDS : Pour une valeur de la charge CPU strictement inférieure à 50%, l'étape du sondage est effectuée sur *N1*. Pour une valeur de la charge CPU supérieure ou égale à 50%, l'étape du sondage est effectuée sur *N2*.

Dans le cas (i), les valeurs des paramètres de la fonction de décision de l'agent restent inchangées pendant l'exécution et donc l'agent reste sur *N1* (inégalité (A)). Dans le cas (ii), les valeurs des paramètres sont calculées pendant la phase de décision et l'agent décide de migrer sur *N2* à partir d'une valeur de la charge de *N1* supérieure ou égale à 50%. Le fait de charger la CPU de *N1* augmente le temps nécessaire pour l'étape du sondage. Alors, l'agent décide de migrer sur *N2* où la CPU est moins chargé pour effectuer cette étape. Puis, le résultat est envoyé sur *N1*où il sera matérialisé.

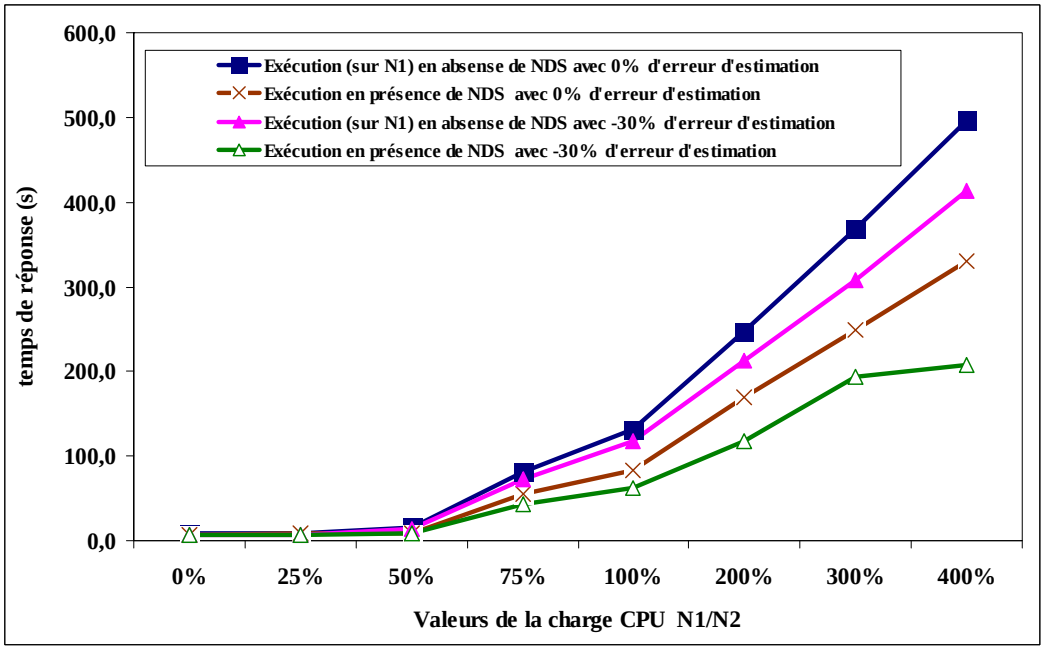

Figure 26 : Temps de réponse en fonction de la variation des valeurs de la charge CPU *N1*/*N2*

Exécution avec des erreurs d'estimation de -60% et -90% sur ||R1||

Symétriquement (par rapport à ce qui est présenté dans la section précédente), on chargera *N2* de X% par rapport à *N1* pour des erreurs d'estimation sur ||*R1*|| comprises entre -60% et -90%.

La Figure 27 présente le temps de réponse de *J* en absence et en présence du NDS avec -60% d'erreur d'estimation sur ||*R1*|| (resp. avec -90% d'erreur d'estimation sur ||*R1*||) en fonction de la variation des valeurs de la charge CPU de *N2*/*N1* de 0% à 400%. Nous avons observé deux comportements différents en absence et en présence du NDS.

(i) Exécution en absence du NDS : Dans ces expériences, quelque soit la variation des valeurs de la charge CPU, l'étape du sondage est effectuée sur *N1*.

(ii) Exécution en présence du NDS : Pour une variation de la charge CPU strictement inférieure à 50%, l'étape du sondage est effectuée sur *N2*. Pour une charge CPU supérieure ou égale à 50%, l'étape du sondage est effectuée sur *N1*.

Dans le cas (i), les valeurs des paramètres de la fonction de décision de l'agent sont constantes durant l'exécution. L'agent décide alors de rester sur *N1*(Inégalité (B)). Dans le cas

(ii), l'agent décide de rester sur *N1* à partir d'une valeur de la charge CPU de *N2* supérieure ou égale à 50%. Le fait de charger la CPU de *N2* augmente le temps nécessaire pour l'étape du sondage. Alors l'agent décide de rester sur *N1* (où la valeur de la charge CPU est moins importante) pour effectuer cette étape et le résultat sera directement matérialisé sur *N1*.

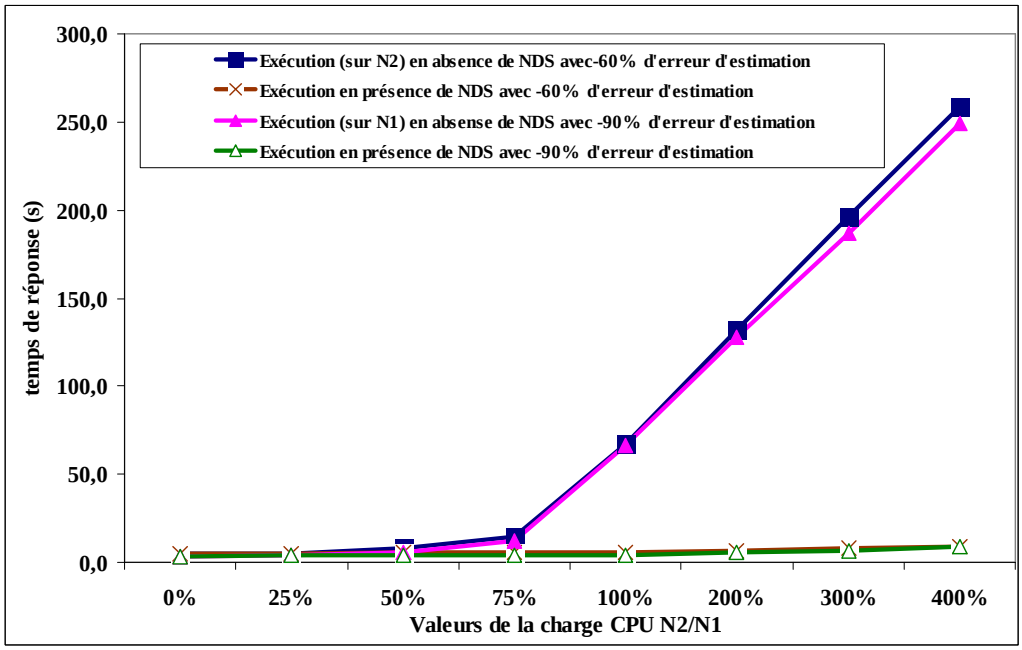

Figure 27 : Temps de réponse en fonction de la variation des valeurs de la charge CPU *N2/N1*

Facteur d'accélération en fonction de la variation des valeurs de la charge CPU

La Figure 28 présente le facteur d'accélération en fonction de la charge CPU des cas précédents (i.e. 0%, -30%, -60% et -90% d'erreurs d'estimation). Ainsi, on constate que l'intérêt de la présence d'un service de monitoring est réel à partir d'une valeur de la charge CPU supérieure ou égale à 50%. La justification de la différence des valeurs des facteurs d'accélération pour des erreurs d'estimation comprises entre 0% et -30% (cas (i), facteur compris entre 1,4 et 1,9) et des erreurs comprises entre -60% et -90% (cas (ii), facteur compris entre 12 et 30) est expliquée dans ce qui suit. Dans le cas (i), le fait de charger *N1* va augmenter le temps de construction de la table de hachage, et la matérialisation dans tous les cas (i.e. exécution de *J* sur *N1* ou sur *N2*). Dans le cas (ii), le fait de charger *N2* va augmenter le temps de traitement de l'étape du sondage seulement si l'exécution est effectuée sur *N2*. Par contre, le temps de la

construction de la table de hachage et de la matérialisation ne sera pas affecté car ces deux étapes sont toujours exécutées sur *N1* d'après l'algorithme de la jointure mobile.

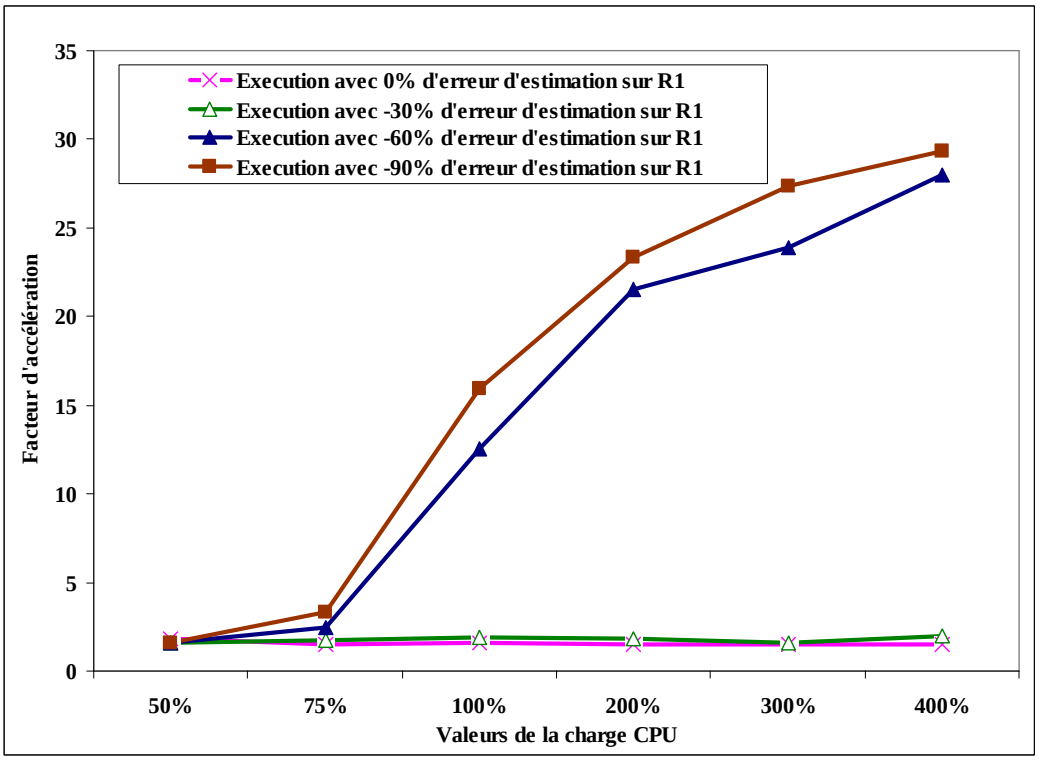

Figure 28 : Facteur d'accélération en fonction de la variation des valeurs de la charge CPU

3.4.2. Impact de la variation des valeurs de la bande passante sur le temps de réponse

Dans cette partie, nous souhaitons étudier l'impact de V2 sur le temps de réponse de *J*. NDS sera utilisé uniquement pour le calcul des valeurs de la charge de la bande passante. Le paramètre CPU sera calibré afin d'isoler chaque paramètre comme nous avons indiqué précédemment.

Exécution avec des erreurs d'estimation de 0%, -30%, -60% et -90% sur ||R1||

La bande passante observée vaut 10 MO/seconde entre *N1* et *N2*. Ensuite, la bande passante est chargée en envoyant des fichiers de *N1* à *N2*. Les valeurs de la charge de la bande passante sont de 25% (7,5 MO/seconde), 50% (5 MO/seconde), 75% (2,5 MO/seconde) et 90% (1 MO/seconde). On n'a pas voulu aller à des valeurs plus petites que 1 MO/seconde car ces valeurs

seront mises en évidence dans les mesures à grande échelle. Dans ce cas, la bande passante est beaucoup plus petite que 1 MO/seconde.

La Figure 29 présente le temps de réponse de *J* en l'absence et en la présence du NDS avec -30% d'erreur d'estimation sur ||*R1*|| et en fonction de la variation des valeurs de la bande passante. Nous avons observé deux comportements différents pour l'exécution en l'absence et en la présence du NDS.

(i) Exécution en absence du NDS : Dans ces expériences, quelque soit les valeurs de la bande passante, l'étape du sondage est effectuée sur *N1*.

(ii) Exécution en présence du NDS : Pour une charge de la bande passante strictement inférieure à 50%, l'étape du sondage est effectuée sur *N1*. Pour une charge de la bande passante supérieure ou égale à 50%, l'étape du sondage est effectuée sur *N2*.

L'interprétation du cas (i) est donnée par l'inégalité (A). Dans ce cas, la fonction de décision ne prend pas en considération la variation des valeurs de la charge de la bande passante. L'interprétation du cas (ii) est la suivante. Si l'on considère l'inégalité (A), *J* doit être exécuté sur *N1* pour n'importe quelle valeur de la charge de la bande passante. Or, pour une erreur d'estimation de -30% sur *R1*, il s'agit du point de basculement de *J* (exécution initiale sur *N1* puis on migre sur *N2* [Hus05a]). C'est le point le plus délicat où le changement du comportement de *J* peut avoir lieu. D'où le résultat.

Nous remarquons aussi que le temps de réponse en présence du NDS est légèrement supérieur au temps de réponse en absence du NDS pour une charge de la bande passante de 0% et 25%. Cela s'explique par le coût d'observation du NDS.

Pour des erreurs d'estimation sur ||*R1*|| de 0%, - 60% et - 90%, on a eu le même comportement pour *J* en présence et en absence du NDS. En effet, le fait d'incrémenter la valeur de la charge de la bande passante n'a aucune influence sur le choix du nœud d'exécution dans ce cas vu que le temps d'envoyer des données entre deux nœuds est proportionnel à la valeur de la bande passante entre ces deux nœuds.

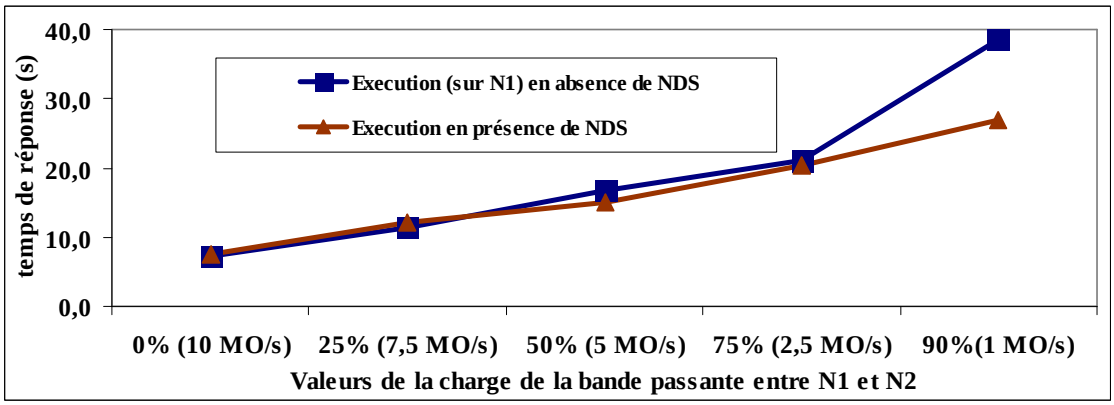

Figure 29 : Temps de réponse en fonction de la variation des valeurs de la bande passante entre *N1* et *N2*

Facteur d'accélération en fonction de la variation des valeurs de la bande passante

Le facteur d'accélération (Figure 30) est présenté uniquement pour le cas d'une erreur d'estimation de -30% sur ||*R1*||. Le service de monitoring s'avère utile à partir d'une bande passante chargée de 50%, en environnement local.

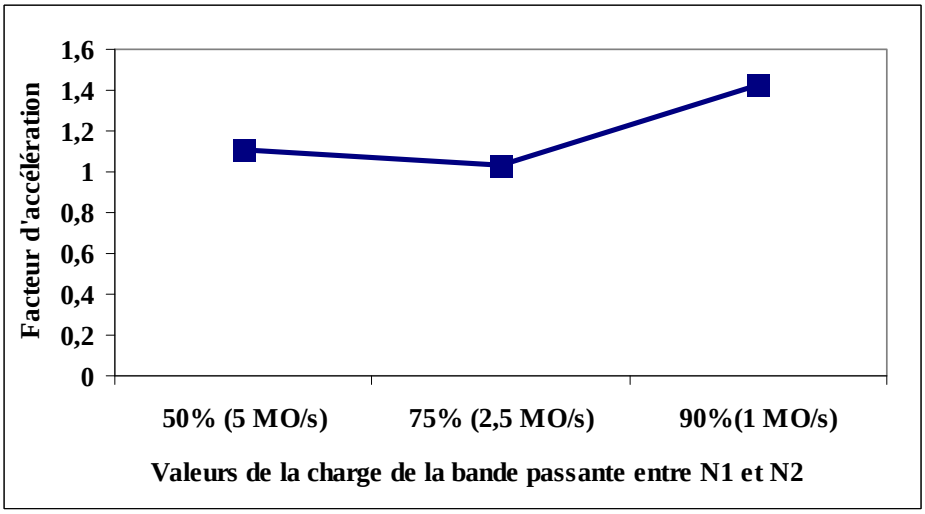

Figure 30 : Facteur d'accélération en fonction de la variation des valeurs de la bande passante entre *N1* et *N2*

3.5. Expériences en environnement à grande échelle

On reprendra les mêmes expériences (avec les mêmes hypothèses) effectuées en environnement local mais cette fois en environnement à grande échelle.

3.5.1. Impact de la variation des valeurs de la charge CPU sur le temps de réponse

Exécution avec des erreurs d'estimation de 0% et -30% sur ||R1||

Nous avons eu les mêmes résultats pour le paramètre CPU en environnement à grande échelle (i.e. 0%, -30%, -60% et -90%) avec une différence au niveau de la valeur de la charge CPU (ici, il s'agit d'une valeur de 75% contre 50% en environnement local). Nous présentons uniquement les résultats en fonction de la variation des valeurs de la charge CPU dans la Figure 31 et dans la Figure 32. L'interprétation de ces résultats est identique à celle présentée en environnement local.

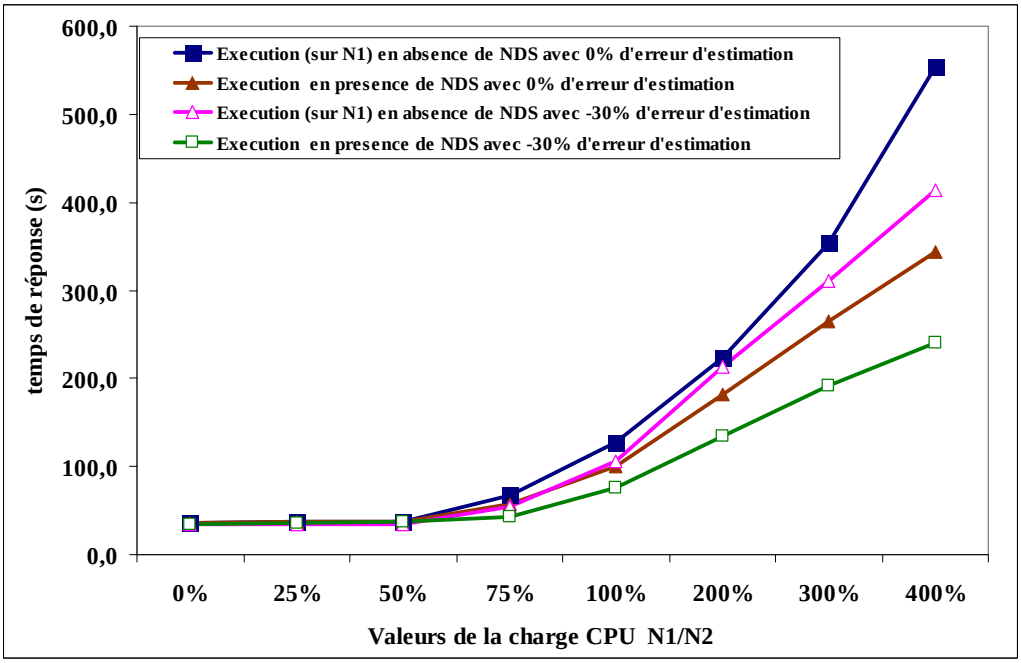

Figure 31 : Temps de réponse en fonction de la variation des valeurs de la charge CPU *N1/N2*

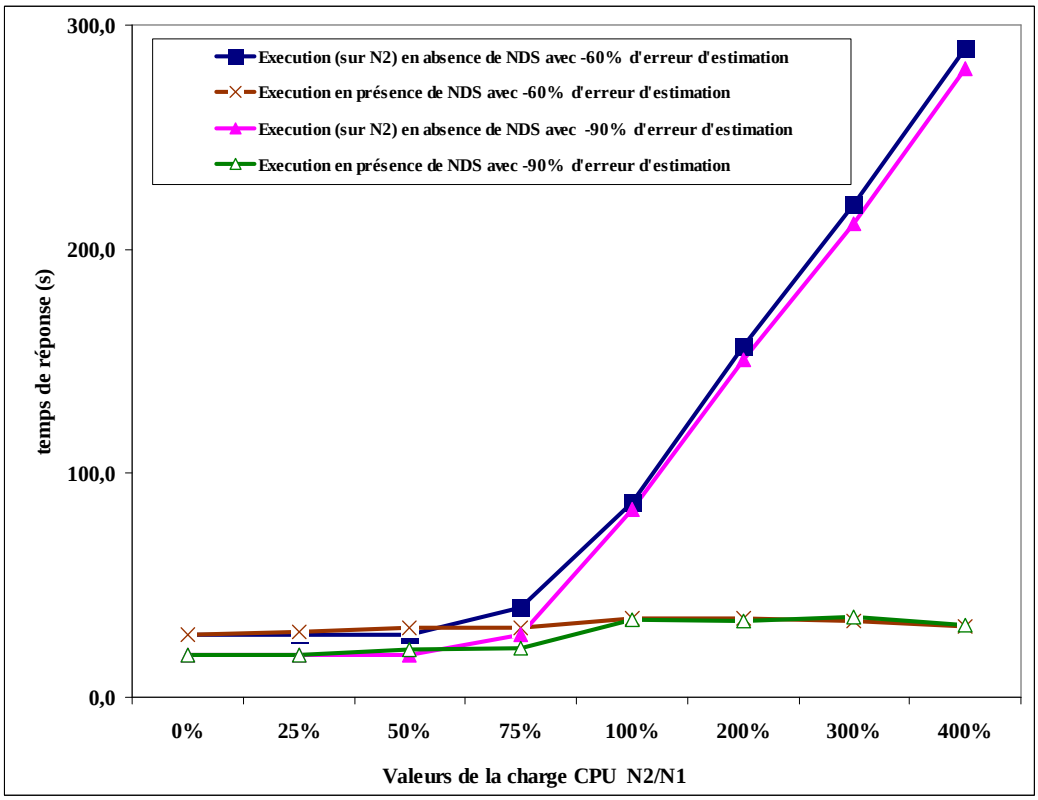

Figure 32 : Temps de réponse en fonction de la variation des valeurs de la charge CPU *N2/N1*

Facteur d'accélération en fonction de la variation des valeurs de la charge CPU

La Figure 33 présente le facteur d'accélération en fonction de la valeur de la charge CPU des cas précédents (i.e. 0%, -30%, -60% et -90% d'erreurs d'estimation). Ainsi, on constate que l'intérêt de la présence d'un service de monitoring est perceptible à partir d'une valeur de la charge CPU qui vaut 75%. L'explication de la différence des valeurs des facteurs d'accélération pour des erreurs d'estimation comprises entre 0% et -30% (cas (i), facteur compris entre 1,2 et 1,7) et pour des erreurs entre -60% et -90% (cas (ii), facteur compris entre 2,4 et 8,7) est identique à celle présentée en environnement local.

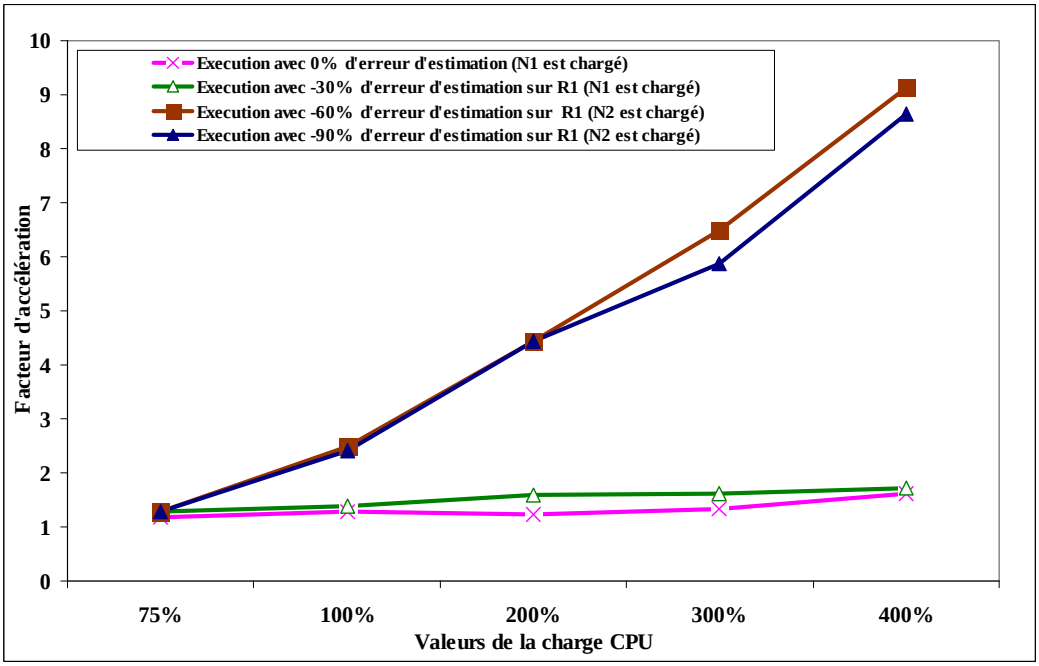

Figure 33 : Facteur d'accélération en fonction de la variation des valeurs de la charge CPU

3.5.2. Impact de la variation des valeurs de la bande passante sur le temps de réponse

Exécution avec des erreurs d'estimation de 0%, -30%, -60% et -90% sur ||R1||

La bande passante observée est de 450 KO/seconde entre *N1* (Toulouse) et *N2* (Lyon). Nous avons chargé la bande passante entre *N1* et *N2* de 25% (350 KO/seconde), 50% (225 KO/seconde) et 75% (120 KO/seconde).

La Figure 34 (resp. la Figure 35) présente le temps de réponse de *J* en l'absence et en la présence du NDS avec 0% d'erreur d'estimation sur ||*R1*|| (resp. avec -30% d'erreur d'estimation sur ||*R1*||) et en variant la valeur de la charge de la bande passante. Nous avons observé deux comportements différents pour l'exécution de *J* en l'absence et en la présence du NDS.

(i) L'exécution en absence du NDS : Dans ces expériences, quelque soit la variation des valeurs de la charge de la bande passante, l'étape du sondage est effectuée sur *N1*.

(ii) L'exécution en présence du NDS : Pour une valeur de la charge de la bande passante strictement inférieure à 25%, l'étape du sondage est effectuée sur *N1*. Pour une charge de la bande passante supérieure ou égale à 25%, l'étape du sondage est effectuée sur N2.

Le comportement des deux courbes de la Figure 34 (resp. la Figure 35) avec 0% d'erreurs d'estimation sur ||*R1*|| (resp. avec -30% d'erreurs d'estimation) doit être théoriquement similaire (à cause de l'inégalité (A)). L'explication de la différence du comportement entre les deux courbes vient de l'asymétrie de la bande passante entre *N1* et *N2*. Durant les expériences menées, la bande passante entre Toulouse et Lyon était différente de celle entre Lyon et Toulouse. L'interprétation des deux cas est donnée par ces deux inéquations:

- Exécution sur *N1* si: $|R1|/BP_{12} + |T|/BP_{21} > |R2|/BP_{21}$ **(C)**
- Exécution sur *N2* si : $|R1|/BP_{12} + |T|/BP_{21} < |R2|/BP_{21}$ **(D)**

Avec BP_{ij} = valeur estimée de la bande passante entre *Ni* et *Nj* ;

Pour des erreurs d'estimation sur ||*R1*|| de -60% et -90%, on a eu le même comportement que *J* en l'absence et en la présence du NDS. L'interprétation est identique à celle présentée en environnement local.

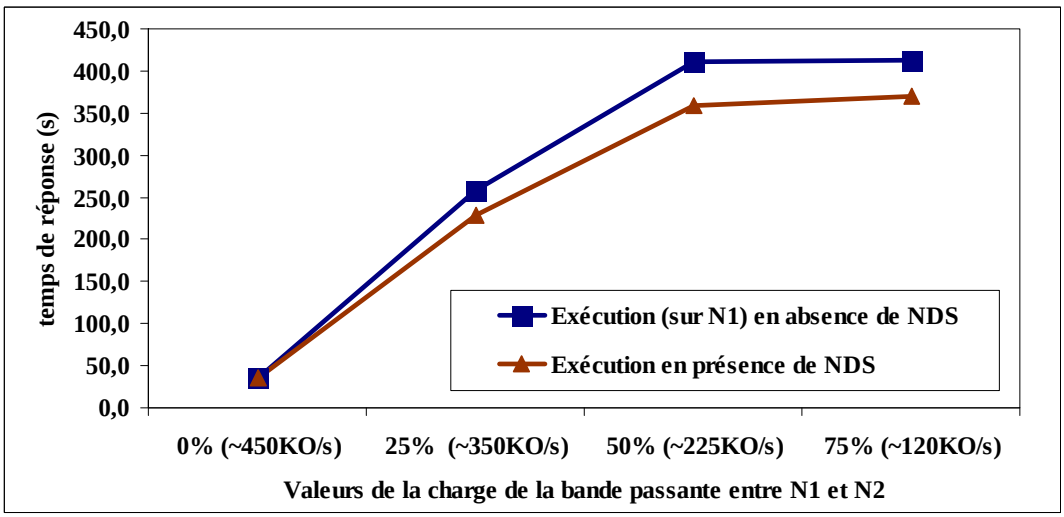

Figure 34 : Temps de réponse en fonction de la variation des valeurs de la bande passante entre *N1* et *N2*

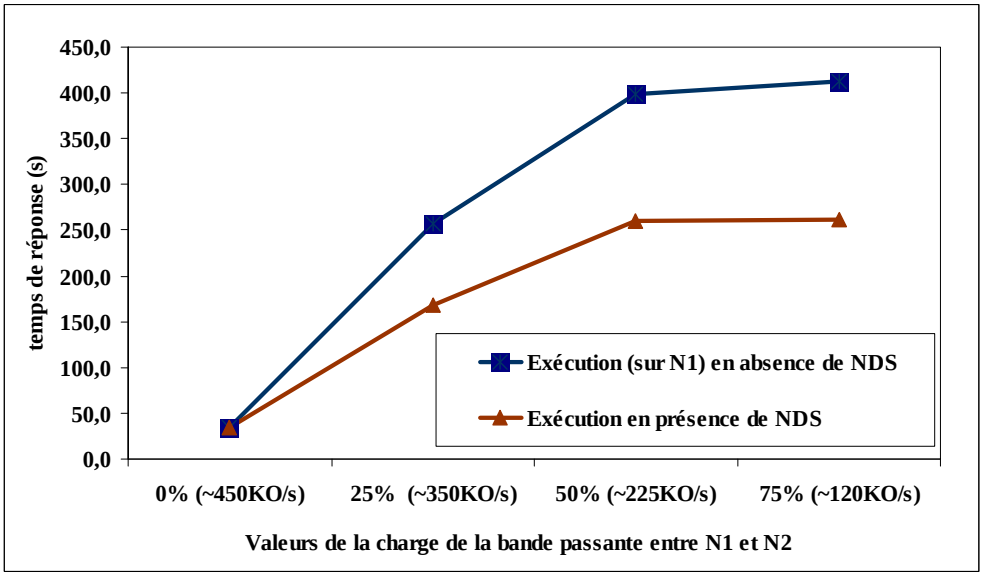

Figure 35 : Temps de réponse en fonction de la variation des valeurs de la bande passante entre *N1* et *N2*

Facteur d'accélération en fonction de la variation des valeurs de la charge de la bande passante

Le facteur d'accélération (Figure 36) est présenté uniquement pour le cas des erreurs d'estimation de 0% et -30% sur ||*R1*||. Le service de monitoring s'avère utile à partir d'une bande passante chargée de 25% dans un environnement à grande échelle.

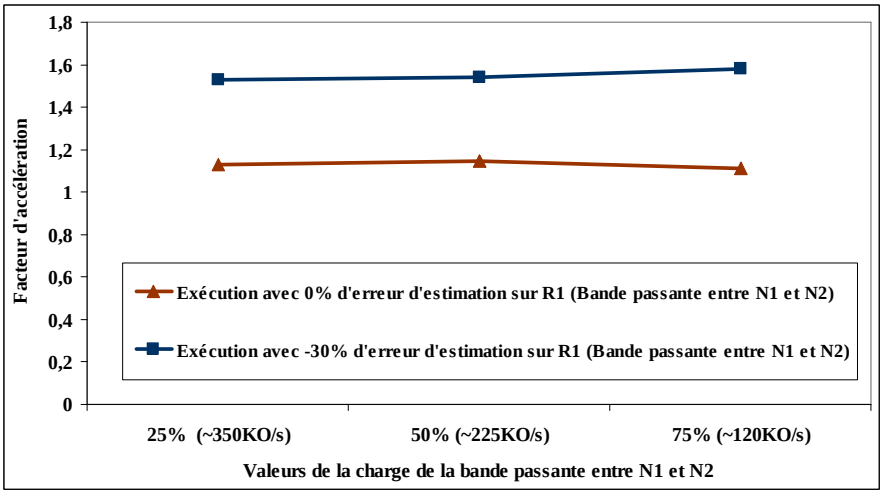

Figure 36 : Facteur d'accélération en fonction de la variation des valeurs de la charge de la bande passante

4. Discussion

4.1. Synthèse des résultats de l'évaluation de la méthode de découverte de relations PTHD

Par rapport au nombre de messages de découverte par seconde

L'évaluation des performances a montré que la méthode SP est meilleure que la méthode PTHD si le nombre de messages de découverte est inférieur à 20 messages de découverte par seconde. Dans le cas contraire, la méthode PTHD est meilleure. Le facteur d'accélération de la méthode PTHD par rapport à la méthode SP est d'environ 5 pour un débit de messages de 0,1 par nœud dans une OV constituée en moyenne de 1000 nœuds. Ainsi, la méthode PTHD permet un meilleur passage à l'échelle que la méthode SP.

D'autre part, la méthode STHD est meilleure que la méthode PTHD lors d'une découverte inter-OV. La méthode PTHD provoque alors un surcoût pour le temps moyen de réponse. Lors d'une découverte intra-OV, la méthode PTHD permet de réduire le coût moyen de la découverte d'une relation par rapport à une méthode STHD.

Par rapport au nombre de nœuds qui se connectent ou qui se déconnectent

La méthode PTHD permet de réduire considérablement le nombre de messages échangés nécessaire à la maintenance du système lors de la connexion ou de la déconnexion d'un nœud et en présence d'un effet churn, par rapport à la méthode STHD. Le facteur d'accélération est compris entre [1,77 ; 2] pour un nombre de nœuds qui se connectent ou qui se déconnectent compris entre 1 et 20.

4.2. Synthèse des résultats de l'évaluation de l'opérateur de jointure mobile par hachage en présence de NDS

4.2.1. Identification des intervalles d'efficacité de l'utilisation de NDS

Par rapport à la valeur de la charge CPU

L'évaluation des performances a montré des facteurs d'accélération très intéressants en fonction de la variation des valeurs de la charge CPU pour toutes les erreurs d'estimation comprises dans l'intervalle [-90% ; 0%]. Dans un environnement local, les facteurs d'accélération

sont compris entre 1,4 (pour une valeur de charge CPU égale à 50%) et 30 (pour une valeur de la charge CPU égale à 400%). Dans un environnement à grande échelle, les facteurs d'accélération sont moins importants (toutefois intéressants) : ils sont compris entre 1,2 (pour une valeur de la charge CPU supérieure ou égale à 75%) et 9,1 (pour une valeur de la charge CPU supérieure ou égale à 400%). Ces différences de valeurs s'expliquent par le fait que le coût de transfert des données en environnement à grande échelle est plus important que le coût d'envoi de ces données en environnement local. D'après ces résultats, nous concluons de l'importance d'observer les valeurs de la charge CPU pour une jointure mobile lorsque les erreurs d'estimation augmentent.

Par rapport à la valeur de la charge de la bande passante

L'évaluation des performances a montrée que le fait de charger la bande passante –même avec de petites valeurs – provoque toujours un facteur d'accélération inférieur à 2 pour toutes les erreurs d'estimation comprises entre l'intervalle [30% ; 0%]. Dans un environnement local, le facteur d'accélération est compris entre 1,03 (pour une valeur de la charge de la bande passante de 50%) et 1,4 (pour une valeur de la charge de la bande passante de 90%). Dans un environnement à grande échelle, le facteur d'accélération est compris entre 1,1 (pour une valeur de la charge de la bande passante de 25%) et 1,6 (pour une valeur de la charge de la bande passante de 75%). Le coût d'exécution doit prendre en considération le coût d'observation de NDS (surtout en environnement local). D'après ces résultats, nous concluons que le bénéfice de l'observation des valeurs de la charge de la bande passante est limité pour une jointure mobile lorsque les erreurs d'estimation augmentent. Un agent mobile peut ainsi embarquer une valeur calibrée pour la bande passante au lieu d'utiliser une méthode capable d'invoquer un outil de monitoring pour le calcul de cette valeur.

5. Conclusion

A partir des simulations évaluant la méthode de découverte de relations, nous avons pu constater que notre méthode PTHD est meilleure que les méthodes STHD lorsque les relations découvertes résident dans l'OV locale. Dans le cas contraire, les méthodes STHD sont meilleures. La méthode est aussi meilleure que les méthodes SP si le nombre de messages dépasse les 20 messages par seconde, i.e. un débit de 0,1 messages par seconde et par nœud dans une OV constituée de 1000 nœuds. Le coût de maintenance des THDs est réduit

considérablement dans la méthode PTHD par rapport aux méthodes STHD en présence d'un effet churn.

Quant à l'évaluation des performances de la jointure mobile, nous avons étudié l'impact de la variation des valeurs de la charge CPU et de la bande passante sur la migration des agents mobiles. Pour mener cette étude, nous avons utilisé NDS. Ce dernier a permis de calculer la meilleure valeur estimée d'un paramètre hôte ou réseau. Dans l'évaluation des performances à grande échelle, nous avons pu constater des facteurs d'accélération compris entre 1,2 et 9,1 par rapport à l'impact de la CPU (pour des valeurs de la charge CPU comprises entre 75% et 400%) pour toutes les erreurs d'estimation comprises entre [-90% ; 0%]. Par contre, le facteur d'accélération ne dépasse la valeur de 2 par rapport à l'impact de la CPU et uniquement pour des erreurs d'estimation comprises entre [-30% ; 0%]. Par conséquent, il est très important d'observer la valeur de la charge CPU pour une jointure mobile lorsque, les erreurs d'estimation augmentent, alors que le bénéfice d'observer la valeur de la charge de la bande passante n'est pas très intéressant dans ce cas. Il devient ainsi opportun d'embarquer dans un agent une méthode capable d'invoquer un outil de monitoring pour capturer la valeur d'un paramètre à jour de la charge CPU d'un nœud.

Chapitre VI. Conclusion et Perspectives

1. Conclusion

Dans cette thèse, nous nous sommes intéressés à la problématique de la découverte de ressources pour l'évaluation de requêtes en environnement de grille de données. Nous avons distingué deux types de ressources : les sources de données (e.g. relations, fichiers XML) et les ressources de calcul (e.g. CPU, mémoire, bande passante). Nous nous sommes focalisés sur la découverte de sources de données [Sam09] qui est spécifique aux grilles de données vu que la découverte de ressources de calcul a été largement abordé dans les grilles de calcul [Iam01, Che05, And02, Cai03, Bha05, Gao04, Jea08, Spe03, Ram04, Sch03, Opp04, …etc.]. Nous avons aussi proposé une stratégie simple permettant la découverte de ressources de calcul [Sam07]. Une fois découvertes, les ressources de calcul nécessitent une étape de monitoring [Sam08] pour observer leur état et ainsi pouvoir réagir en fonction de la variation des valeurs des paramètres hôtes (i.e. valeurs de la charge CPU, la charge des E/S et la charge mémoire d'un nœud) et réseaux (i.e. valeurs de la latence et de la charge de la bande passante entre deux nœuds).

Nous avons abordé dans le chapitre II, le problème de la découverte de ressources d'une manière générale et nous avons défini un ensemble de critères majeurs à considérer pour la comparaison des méthodes de découverte de ressources. Nous avons ensuite spécifié la particularité de la découverte de sources de données par rapport à la découverte de ressources de calcul. Outre la prise en compte des caractéristiques de la grande échelle et de l'instabilité du système lors de la conception d'une méthode de découverte de ressources, il est fondamental qu'une méthode de découverte de sources de données soit fiable, i.e. la découverte doit indiquer, pour toutes les sources de données référencées dans une requête SQL, si ces sources se trouvent bien dans le système. Auquel cas, la méthode doit indiquer que ces sources ne sont pas connectées au système.

Dans le chapitre III, nous avons proposé une méthode [Sam09] de découverte de sources de données pour l'évaluation de requêtes en environnement de grille de données. L'idée est de décomposer la grille en plusieurs OVs [Fos01, Iam01, Mas05, Tal05]. Ensuite, nous proposons d'associer une THD_i pour chaque OV_i. Chaque THD_i est responsable de stocker les métadonnées décrivant toutes les sources de données stockées dans une OV_i. Lorsqu'un utilisateur soumet une requête de découverte, la découverte intra-*OV* est d'abord lancée dans l'OV_{locale} à laquelle appartient l'utilisateur, car nous favorisons le principe de la localité de données [Wat05]. La découverte intra-*OV* est une découverte classique basée sur les THDs qui peut passer à l'échelle et fonctionner en présence de la dynamicité des nœuds. Si les relations à découvrir ne se trouvent pas dans l'OV_{locale}, alors il faut propager la requête de découverte vers toutes les autres OV_i $(i \neq locale)$. Pour pouvoir interconnecter efficacement les OVs, nous proposons un système d'adressage permettant un accès permanent d'un nœud *N* d'une OV_{locale} vers toute autre OV_i ($i \neq locale$). *N* connaît un et un seul point de sortie (i.e. un nœud) N_i vers une OV_i. Les points de sortie vers une OV_i ne sont pas tous identiques afin d'éviter les inconvénients des nœuds centraux (i.e. le goulet d'étranglement si un grand nombre de messages est soumis à un nœud central ; la panne de ce nœud peut alors causer le dysfonctionnement d'une partie du système). Le système d'adressage possède un faible coût de maintenance en présence de l'effet churn grâce à l'emploi d'une politique de mise à jour paresseuse. La mise à jour de ce système est effectuée lors de la découverte inter-OV.

Dans le chapitre IV, nous avons intégré les informations de monitoring dans un modèle d'exécution à base d'agents mobiles développé au sein de l'équipe pyramide [Arc04, Ham06, Hus05a, Mor03]. Nous avons proposé une méthode [Sam08] permettant de prendre en considération la variation des valeurs des paramètres hôtes et réseaux dans le calcul du coût d'exécution d'une opération relationnelle. Plus précisément, la méthode proposée permet à un agent mobile de capturer la meilleure valeur estimée d'un paramètre hôte ou réseau grâce à l'utilisation des informations de monitoring pendant l'exécution. La méthode proposée améliore nettement la qualité des décisions de l'optimiseur.

Pour valider nos propositions, nous avons développé un modèle de simulation car nous ne disposons pas d'accès à un grande nombre de nœuds (qui peut atteindre des milliers voire une dizaine de milliers) pour étudier le comportement de notre méthode de découverte de ressources

(de type sources de données). En se basant sur ce modèle de simulation, nous avons comparé les performances de notre méthode utilisant Plusieurs THDs (PTHD), par rapport aux méthodes les plus employées dans la littérature pour la découverte de ressources : les méthodes basées sur des techniques Super-Pairs (SP) [Mas05, Pup05] et celles basées sur l'utilisation d'une Seule THD dans tout le système (STHD) [Dov03, Gar03, Row01, Sto01]. Les ressources considérées dans les trois méthodes sont des relations. A partir de ces simulations, nous constatons que :

(i) Pour un débit de message inférieur à une vingtaine de messages de découverte par seconde, les méthodes SP sont meilleures que la méthode PTHD, en terme de temps moyen de réponse. Sinon, la méthode PTHD est meilleure que les méthodes SP. Le facteur d'accélération calculé est d'environ 5 pour un débit de message de découverte de 0,1 par nœud dans une OV constituée en moyen de 1000 nœuds,

(ii) Lorsque les relations à découvrir résident dans une OV distante, les méthodes STHD sont meilleures que la méthode PTHD, en terme de temps moyen de réponse, d'environ 17% pour un débit de message de découverte de 0,1 par nœud dans une OV constituée en moyenne de 1000 nœuds. Sinon, la méthode PTHD est meilleure que les méthodes STHD avec un gain (sur le temps moyen de réponse) d'environ 13%,

(iii) La méthode PTHD réduit considérablement le nombre de messages échangés dans le système par rapport aux méthodes STHD en présence de la connexion et de la déconnexion des nœuds. Le facteur d'accélération calculé, en terme de nombre de messages échangés dans le système, est compris entre 1.77 (pour 1 nœud connecté ou déconnecté) et 2.04 (pour un effet churn de 20 nœuds connectés ou déconnectés).

Quant à l'évaluation des performances de la jointure mobile en présence de NDS, nous nous sommes appuyés sur une plate-forme d'expérimentation d'agents mobiles [Arc02], le langage JAVA et l'intergiciel RMI. En utilisant cette plate-forme d'expérimentation, nous avons mesuré, par exécution, les temps de réponse de la jointure mobile par hachage (J : T= $R1 \infty R2$) en présence de NDS en environnement local et en environnement à grande échelle. A partir de ces simulations, nous constatons que :

(i) Les temps de réponse de la jointure mobile par hachage en présence de NDS sont meilleurs que ceux de la jointure mobile par hachage en absence de NDS pour une valeur de la charge CPU > 50% (resp. >75%) en environnement local (resp. en environnement à grande échelle) pour toutes les erreurs d'estimation de *R1* comprises entre [-90% ; 0%]. Les facteurs d'accélération sont compris entre 1,4 et 30,

(ii) Les temps de réponse de la jointure mobile par hachage en présence de NDS sont meilleurs que ceux de la jointure mobile par hachage en absence de NDS pour une valeur de la charge de la bande passante > 50% (resp. > 25%) en environnement local (resp. en environnement à grande échelle) pour toutes les erreurs d'estimation de *R1* comprises entre [-30% ; 0%]. Les facteurs d'accélération sont compris entre 1,1 et 1,6.

A partir de ces constatations, il est capital d'observer les valeurs de la charge CPU quant à l'observation des valeurs de la charge de la bande passante est moins importante pour l'opérateur de la jointure mobile par hachage [Arc04] en présence des erreurs d'estimation.

2. Perspectives

Dans nos futurs travaux de recherche, nous souhaiterons l'extension de notre méthode de découverte de sources de données à n'importe quel type de ressource (e.g. services, programmes). Nous voudrons valider et évaluer les propriétés de notre méthode de découverte de ressources par exécution en utilisant une réelle plateforme d'expérimentation comme la grille 5000 [Gri09]

Dans un second temps, nous souhaiterons étudier, d'une manière plus profonde, le comportement de notre méthode de découverte de ressources en présence d'un effet churn, en modélisant l'arrivée et le départ des nœuds suivant des fonctions prédéfinies. Par exemple, les auteurs dans [Wu06, Stu05] associent pour chaque nœud, une durée de vie ou une durée de session. Ainsi, les nœuds quittent le système lorsque leur durée de vie écoule. Plusieurs distributions pour la durée de vie d'un nœud peuvent être considérées : une distribution exponentielle [Wu06], une distribution de Pareto observée dans les vrais systèmes P2P [Sar04], une distribution cumulative [Stu05], …etc. Ainsi, il devient intéressant de modéliser le processus d'arrivée des nœuds et d'étudier son impact sur la méthode de découverte de ressources [Sam09].

Références bibliographiques

[Abd05] A. Abdullaha et al.; *Data discovery algorithm for scientific data grid environment*, J. Parallel Distributed Computing, pp.1429–1434, Elsevier 2005.

[Ada96] S. Adali et al. ; *Query Caching and Optimization in Distributed Mediator Systems*, Proc. of ACM SIGMOD International Conference on Management of Data, ACM Press, Montreal, Canada, pp. 137-148, june1996,.

[And02] A. Andrzejak , Z. Xu, *Scalable Efficient Range Queries for Grid Information Services*, Proceedings of the Second International Conference on Peer-to-Peer Computing, p.33, September 05-07, 2002.

[Akb07] R. Akbarinia, V. Martins; *Data Management in the APPA System*, Journal of Grid Computing, Springer,Vol. 5, Number 3, pp. 303-317, 2007.

[Ali07] H. A. Ali; *A Framework for Scalable Autonomous P2P Resource Discovery for the Grid Implementation*, International Journal of Computer Science and Engineering, PWASET, Vol.25, 2007.

[Alo03] Aloisio et al.; *The Grid-DBMS: Towards Dynamic Data Management in Grid Environments*, Int. Conf. on Information Technology: Coding and Computing, pp. 199 – 204, 2005.

[Alp03] M. N. Alpdemir et al.; *Service based distributed querying on the grid*. In Proc. of ICSOC, LNCS, pages 467–482. Springer, 2003

[Alp05] M. N. Alpdemir et al.; *Using OGSA-DQP to Support Scientific Applications for the Grid*, Proc. of SAG 2004, LNCS 3458, pp. 13-24, 2005.

[Al-H05] L. Al-Hussaini et al.; *A Unified Grid Query Service for Grid Resource Brokers*, ITCC : 760-761, 2005.

[Ant05] M. Antonioletti et al.; *The design and implementation of Grid database services in OGSA-DAI*, Concurrency and Computation: Practice & Experience, Vol. 17, pp. 357–376, 2005.

[Arc04] J.-P. Arcangeli et al.; *Mobile Agent Based Self-Adaptive Join for Wide-Area Distributed Query Processing*, Journal of Database Management, Idea Group, Vol. 15, N. 4, pp. 25-44, 2004.

[Bha05] Bhatia, K.: OGSA-P2P research group; *peer-to-peer requirements on the open Grid services architecture framework*, In: Global Grid Forum Document GFD-I.049, 2005.

[Avn00] R. Avnur, J. Hellerstein. Eddies; *continuously adaptive query processing*, In Proc. of ACM SIGMOD 2000, pp. 261–272, 2000.

[Bal01] Z. Balaton et al.; *Use Cases and the Proposed Grid Monitoring Architecture*, Tech. Rep. LDS-1/2001, Computer and Automation Research Institute of the Hungarian Academy of Sciences, http://www.lpds.sztaki.hu/publications/reports/lpds-1-2001.pdf, 2001.

[Bos05] S. K. Bose et al.; *Allocating Resources to Parallel Query Plans in Data Grids*, Proc. of the 6th Intl. Conf. on Grid and Cooperative Computing, Vol. 00, pp. 210-220, 2007.

[Cai03] M. Cai et al.; *MAAN: A Multi-Attribute Addressable Network for Grid Information Services*, Proc. of 4th Intl. Workshop on Grid Computing, pp. 184 – 191, 2003.

[Cao02] J. Cao et al.; *Agent-Based Resource Management for Grid Computing*, 2nd IEEE/ACM International Symposium on Cluster Computing and the Grid, Berlin, pp.350-351, 2002.

[Cao01] J. Cao et al.; *Performance Evaluation of an Agent-Based Resource Management Infrastructure for Grid*, Computing, Proceedings of 1st IEEE/ACM International Symposium on Cluster Computing and the Grid. Brisbane, Australia, 311–318, May 2001.

[Che00] A. Chervenak et al.; *The data grid : Towards an architecture for the distributed management and analysis of large scientific datasets*, vol. 23, n°3, pp. 187-200, 2000.

[Che05] S. Cheema et al.; *Peer-to-peer Discovery of Computational Resources for Grid Applications*, Proc. of Grid Computing Workshop, IEEE CS, pp. 179-185, 2005,.

[Coo03] Cooke et al.; *R-GMA: an information integration system for grid monitoring*, Proc. of the 10th Intl. Conf. on Cooperative Information Systems, pp. 462-481, 2003.

[Cza01] K. Czajkowski et al.; *Grid information services for distributed resource sharing*, Proc of the 10th IEEE Symp. on HPDC, IEEE CS, pp. 181-194, 2001.

[Dee04] E. Deelman et al.; *Su: Grid-Based Metadata Services*, SSDBM, pp. 393-402, 2004.

[Din01] P. Dinda and B. Plale; *A unified relational approach to grid information services*, Technical Report GWD-GIS-012-1, Global Grid Forum, 2001.

[Din05] S. Ding et al.; *A Heuristic Algorithm for Agent-Based Grid Resource Discovery*, Intl. Conf. on e-Technology, e-Commerce and e-Service, IEEE, pp. 222-225, 2005.

[Dov03] D. Doval and D. O'Mahony; *Overlay networks: A scalable alternative for P2P*, Internet Computing, IEEE, Vol. 7, pp.79-82, 2003.

[Du95] W. Du, M.-C. Shan ; *Query Processing in Peagasus. In Object-Oriented Multidatabase systems : A Solution for Advanced Applications*, pp. 449-468, 1995.

[Fit97] S. Fitzgerald et al.; *A Directory Service for Configuring High-Performance Distributed Computations*, HPDC pp.365-376, 1997.

[Fil04] Filho et al.; *PerDiS: a scalable resource discovery service for the ISAM pervasive environment*, International Workshop on Hot Topics in Peer-to-Peer Systems, pp. 80-85, Oct. 2004.

[Fos01] Ian T. Foster; *The Anatomy of the Grid: Enabling Scalable Virtual Organizations*, Lecture Notes In Computer Science, Vol. 2150, Euro-Par, Conference on Parallel Processing, Springer-Verlag, pp.1– 4, 2001.

[Fos02] I. Foster; *The Grid: A New Infrastructure for 21st Century Science*, Physics Today, Vol. 55, N. 2, pp. 42-56, 2002.

[Fos04a] I. Foster, N. R. Jennings and C. Kesselman; *Brain Meets Brawn: Why Grid and Agents Need Each Other*, Third International Joint Conference on Autonomous Agents and Multiagent Systems, pp.: 8 – 15, 2004.

[Fos04b] I. Foster; *Globus Toolkit Version 4: Software for Service-Oriented Systems*, IFIP Intl. Conf. on Network and Parallel Computing, LNCS 3779, pp. 2-13, 2005.

[Fuk06] M. Fukuda and D. Smith; *UWAgents: A mobile agent system optimized for grid computing*, in Proc. of the 2006 International Conference on Grid Computing and Applications – CGA'06. Las Vegas, NV: CSREA, pp. 107–113, June 2006.

[Gal03] L. Galanis et al., "Locating data sources in large distributed systems", Int. conf. on Very Large Data Bases, Vol. 29, pp: 874-885, 2003.

[Gao04] J. Gao and P. Steenkiste; *An Adaptive Protocol for Efficient Support of Range Queries in DHT-based Systems*, Proceedings of 12th IEEE International Conference the Network Protocols (ICNP), pp.239-250, IEEE Computer Society, 2004.

[Gar90] G. Gardarin and P. Valduriez ; *SGBD Avancés, Bases de données objets, déductives et réparties*, Eyrolles, 1990.

[Gar03] L. Garces-Erce et al.; *Hierarchical peer-to-peer systems*, In Proc. Euro-Par 2003, Klagenfurt, Austria, 2003.

[Gar96a] G. Gardarin, F. Sha, Z.-H. Tang ; *Calibrating the Query Optimizer Cost Model of IRO-DB, an Object-Oriented Federated Database System*, Proc. of 22th International Conference on Very Large Data Bases, Morgan Kaufmann, Mumbai (Bombay), India, pp. 378-389, september 1996.

[Gri02] GridFTP, http://www.globus.org/grid_software/data/gridftp.php, 2002.

[Gos07a] J. Gossa, J.-M. Pierson; *End-to-End Distance Computation in Grid Environment by NDS, the Network Distance Service*, Fourth European Conference on Universal Multiservice Networks (ECUMN 2007), IEEE Computer Society, 210-222, 2007.

[Gos07b] J. Gossa; *NDS: Network Distance Service*, http://liris.cnrs.fr/~jgossa/nds/, 2007.

[Gou040a] A. Gounaris et al.; *Adaptive Query Processing and the Grid: Opportunities and Challenges*, In Proc. of the 15th Intl. Dexa Workhop, IEEE CS, pp. 506-510, 2004.

[Gou04b] A. Gounaris et al.; *Self-monitoring query execution for adaptive query processing*, Data Knowl. Eng., pp. 325-348, 2004.

[Gou05] A. Gounaris et al.; *Adapting to Changing Resource Performance in Grid Query*, Proc. Of the Intl. VLDB Workshop, Data Management in Grids, LNCS 3836, pp. 30–44, 2005.

[Gou06] A. Gounaris et al.; *A novel approach to resource scheduling for parallel query processing on computational grids*, Distributed and Parallel Databases, pp. 87-106, 2006.

[Gri09] *Grid 5000;* www.grid5000.org, 2009.

[Ham96] A. Hameurlain, P. Bazex, Franck Morvan ; *Traitement parallèle dans les bases de données relationnelles : concepts, méthodes et applications*, Cépaduès Editions, 1996.

[Ham06] A. Hameurlain, F. Morvan; *How Can Mobile Agents and Cost Model Approaches Help for Distributed Query Optimization?*, Proc. of the17th Int. Dexa Workshop, IEEE CS, pp. 617-621, 2006.

[Ham08] A. Hameurlain, F. Morvan, M. El Samad; *Large Scale Data management in Grid Systems: a Survey*, IEEE International Conference on Information and Communication Technologies: from Theory to Applications (ICTTA 2008), IEEE, 2008.

[Ham09] Abdelkader Hameurlain, Deniz Cokuslu, Kayhan Erciyes; *Resource Discovery in Grid Systems: a Survey*, Rapport de recherche, IRIT/RT--2009-3--FR, IRIT, avril 2009.

[Har03] N. Harvey et al.; *Skipnet: A scalable overlay network with practical locality properties*, In Proc. USITS 2003, Seattle, WA, 2003.

[He05] Y. He et al.; *Agent-based Mobile Service Discovery in Grid Computing*, Proceedings of the 2005 The Fifth International Conference on Computer and Information Technology (CIT 2005), IEEE, 2005.

[Hon05] Honeyman, W.A., Adamson, P., McKee, S.: GridNFS: global storage for global collaborations. In: IEEE Int. Symp. on Global Data Interoperability – Challenges and Technologies, pp. 111–115, 2005.

[Hua04] C.Huang et al.; *Dart: A Framework for Grid-Based Database Resource Access and Discovery*, LNCS, Book of Grid and Cooperative Computing, Vol. 3033, pp. 855-862, 2004.

[Hue03] Huebsch R et al,; *Querying the internet with PIER* In: VLDB Conference, 2003.

[Hus05a] M. Hussein; Un modèle d'exécution à base d'agents mobiles pour l'optimisation dynamique de requêtes réparties à grande échelle, Thèse de doctorat, IRIT, France, Décembre 2005.

[Hus05b] M. Hussein et al.; *Embedded Cost Model in Mobile Agents for Large Scale Query Optimization*, Proc. of the 4th Intl. Symposium on Parallel and Distributed Computing, IEEE CS, pp. 199-206, 2005.

[Hus06] M. Hussein et al.; *Dynamic Query Optimization: from Centralized to Decentralized*, 19th Intl. Conf. on Parallel and Distributed Computing Systems, p. 273-279, 2006.

[Iam01] A. Iamnitchi; *On Fully Decentralized Resource Discovery in Grid Environments*, Book Series, Lecture Notes in Computer Science, Springer Berlin / Heidelberg, Vol 2242/2001, Book Grid Computing, pp. 51-62, 2001.

[Iam02] A. Iamnitchi, M. Ripeanu, I. T. Foster; *Locating Data in (Small-World?) Peer-to-Peer Scientific Collaborations*, IPTPS, pp.232-241, 2002.

[Iam04] A. Iamnitchi, I. T. Foster, D. Nurmi; *A Peer-to-Peer Approach to Resource Location in Grid Environments*, Grid resource management: state of the art and future trends, pp.: 413 - 429, 2004.

[Iam05] A. Iamnitchi, D. Talia; *P2P computing and interaction with Grids*, Future Generation Computer Systems, Vol. 21, N. 3, pp. 331–332, 2005.

[Jay04] S. Jayasena et al.; *Data Resource Discovery in a Computational Grid*, Lecture Notes in Computer Science, Springer, Vol. 3252, International Workshop on Storage Grid and Technologies, pp.303-310, 2004

[Jea08] E. Jeanvoine and C. Morin; *RW-OGS: An optimized randomwalk protocol for resource discovery in large scale dynamic Grids*, Grid Computing Conference, IEEE/ACM, pp. 168-175, 2008.

[Jon97] R. Jones, J. Brown; *Distributed query processing via mobile agents*, 2002, accessible via : http://www.cs.umd.edu/~rjones/paper.html, 1997.

[Kak06] G. Kakarontzas, I. K. Savvas; *Agent-Based Resource Discovery and Selection for Dynamic Grids*, Proc of the 15th IEEE Intl. Workshops on Enabling Technologies, pp. 195-200, 2006.

[Kau07] D. Kaur, J. Sengupta; *Resource Discovery in Web-services based Grids*, Proceedings of World Academy of Science, Engineering and Technology, pp.284-288, 2007.

[Kin07] R. Al King, A. Hameurlain, F. Morvan; *Metadata Lookup for Distributed Query Optimization in P2P Environment*, International Conference on Parallel and Distributed Computing Systems (PDCCS 2007), Las Vegas, Nevada, International Society for Computers and their Applications (ISCA), pp. 36-43, 2007.

[Kin09] R. Al King, A. Hameurlain, F. Morvan; *Ontology-Based Data Source Localization in a Structured Peer-to-Peer Environment*, International Database Engineering & Applications Symposium (IDEAS 2008), Coimbra, Portugal, ACM, p. 9-18, 2008.

[Liu08] S. Liu, H.A. Karimi; *Grid query optimizer to improve query processing in grids*, Future Generation Computer Systems, 2008.

[Lyn09] A. Lyn et al.; *The design and implementation of OGSA-DQP: A service-based distributed query processor*, Future Generation Computer Systems, Vol. 25, Issue 3, pp. 224-236, 2009.

[Mar05] M. Marzolla et al.; *Peer-to-Peer for Discovering resources in a Dynamic Grid* , Journal of Parallel Computing, Vol. 33, N. 4-5, pp. 339-358, 2007.

[Mas05] C. Mastroianni, D. Talia, O. Verta; *A super-peer model for resource discovery services in large-scale Grids*, Future Generation Computer Systems, Vol. 21, pp. 1235-1248, Elsevier Science, 2005.

[Mas08] C. Mastroianni, D. Talia, O. Verta; *Designing an information system for Grids: Comparing hierarchical, decentralized P2P and super-peer models*, Parallel Computing, Vol.34, issue 10, pp. 593-611, 2008.

[Mes08] E. Meshkova et al.; *A survey on resource discovery mechanisms, peer-to-peer and service discovery frameworks*, the International Journal of Computer and Telecommunications Networking archive, Vol. 52 , Issue 11, pp. 2097-2128, August 2008.

[Mis04] A. Mislove, P. Druschel; *Providing Administrative Control and Autonomy in Structured Peer-to-Peer Overlays*, pp. 162-172, IPTPS 2004.

[Mor03] F. Morvan, M. Hussein, A. Hameurlain; *Mobile Agent Cooperation Methods for Large Scale Distributed Dynamic Query Optimization.* Dans : International Conference on Database and Expert Systems Applications (DEXA 2003), IEEE Computer Society, pp. 542-547, september 2003.

[Naa99] H. Naacke ; *Modèles de coût pour médiateurs de bases de données hétérogènes*, Thèse de Doctorat, Spécialité Informatique, Université de Versailles Saint-Quentin-en-Yvlenies, France, 1999.

[Opp04] D. Oppenheimer et al.; *Scalable wide-area resource discovery*, TR CSD04-1334, Univ. of California, 2004.

[Ozs99] M. T. Özsu, P. Valduriez; *Principles of Distributed Database Systems,* Second Edition Prentice-Hall, 1999.

[Pup05] D. Puppin et al.; *A grid information service based on peer-to-peer*, EuroPar, Springer LNCS, Volume 3648/2005, pp.454–464, 2005.

[Pas08] S. Pastore*; The service discovery methods issue: A web services UDDI specification framework integrated in a grid environment*, J. Network and Computer Applications, pp. 93-107, 2008.

[Pac07] E. Pacitti et al.; *Grid Data Management: Open Problems and News Issues*; Intl. Journal of Grid Computing; Springer, Vol. 5, N. 3, pp. 273-281, 2007.

[Pie07] J.-M. Pierson et al.; *GGM Efficient Navigation and Mining in Distributed Geno-Medical Data*, IEEE Transactions on Nanobioscience, IEEE, Vol. 6, N. 2, pp. 110-116, June 2007.

[Ram04] S. Ramabhadran et al.; *Prefix Hash Tree: An Indexing Data Structure over Distributed Hash Tables,* PODC, 2004.

[Ran08] R. Ranjan, A. Harwood, R. Buyya; *Peer-to-Peer Based Resource Discovery in Global Grids: A Tutorial*, IEEE Communications Surveys and Tutorials, Vol. 10, No2, pp. 6-33, 2008.

[Ram06] T. G. Ramos et al.; *An Extensible Resource Discovery Mechanism for Grid Computing Environments*, Proc. Of the 6th IEEE Intl. Symposium on Cluster Computing and the Grid, pp. 115-122, 2006.

[Rat01] S. Ratnasamy et al.; *A scalable content-addressable network,* Proceedings of the 2001 conference on Applications, technologies, architectures, and protocols for computer communications, p.161-172, San Diego, California, United States, August 2001.

[Row01] A. Rowstron, P. Druschel; *Pastry: Scalable, distributed object location and routing for large-scale peer-to-peer systems*, Proc. Of the 18th IFIP/ACM Intl. Conf. on Distributed Systems Platforms, Vol. 2218, pp. 329-350, 2001.

[Sam07] M. El Samad, A. Hameurlain, F. Morvan; *Resource Discovery and Selection for Large Scale Query Optimization in a Grid Environment*, International Conference on Systems, Computing Sciences and Software Engineering (SCSS 2007), Springer, Advances in Computer and Information Sciences and Engineering, pp. 279-283, 2007.

[Sam08] M. El Samad et al.; *A monitoring service for large scale dynamic query optimization in a Grid environment*, Intl. Journal of Web and Grid Services (IJWGS), Vol.4, No.2, pp. 226-246, 2008.

[Sam09] M. El Samad, F. Morvan, A. Hameurlain; *Resource Discovery for Query Processing in Data Grids*, Dans : 22nd International Conference on Parallel and Distributed Computing and Communication Systems (PDCCS), Louisville, Kentucky USA, International Society for Computers and their Applications (ISCA), September 2009.

[Sar04] S. Saroiu; *Measurement and analysis of internet content delivery systems*, Doctoral Dissertation, University of Washington, December 2004.

[Smi02] J. Smith et al.; *Distributed Query Processing on the Grid*, GRID 2002, LNCS 2536, pp. 279–290, Springer-Verlag Berlin Heidelberg, 2002.

[Sha08] A. Sharma, S. Bawa; *Comparative Analysis of Resource Discovery Approaches in Grid Computing*, Journal of computers, Vol. 3, No. 5, may 2008.

[Sch03] C. Schmidt and M. Parashar; *Flexible information discovery in decentralized distributed systems*, In HPDC'03: In Proceedings of the Twelfth International Symposium on High Performance Distributed Computing, June, Washington, DC, USA, IEEE, 2003.

[Sch04] J. M. Schopf; *Grid resource management: state of the art and future trends*, Norwell, MA, Kluwer, Academic Publishers, 2004.

[Sch06] J. M. Schopf et al.; *Monitoring the Grid with the Globus Toolkit MDS4*, Invited paper, Journal of Physics: Conference Series, Proceedings of SciDAC 2006, June 2006.

[Sed08] M. Sedaghat , M. Othman and M. N. Sulaiman; *Agent's role in grid environments during resource discovery: A review*, Information Technology, ITSim 2008, International Symposium on. Vol. 3, No. 26-28, pages 1-9, 2008.

[Smi02] J. Smith et al.; *Distributed Query Processing on the Grid*, Proc. of the Third Workshop on Grid Computing, GRID 2002, LNCS 2536, pp. 279–290, 2002.

[Soe05] K. M. Soe et al.; *Efficient Scheduling of Resources for Parallel Query Processing on Grid-based Architecture*, Proc. of the 6th Asia-Pacific Symposium, IEEE, pp. 276-281, 2005.

[Spe03] D. Spence , Tim Harris; *XenoSearch: Distributed Resource Discovery in the XenoServer Open Platform*, Proceedings of the 12th IEEE International Symposium on High Performance Distributed Computing, p.216, June 22-24, 2003.

[Sto01] I. Stoica et al.; *Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications*, Proc. of the ACM SIGCOMM Conf., pp. 17-32, 2001.

[Stu05] D. Stutzbach, R. Rejaie*; Characterizing Churn in Peer*-to-*Peer Networks*, Technical Report CIS-TR-2005-03, University of Oregon. June 3, 2005.

[Tal05] D. Talia, P. Trunfio; *Peer-to-Peer protocols and Grid services for resource discovery on Grids*, Advances in Parallel Computing, Vol. 14, Elsevier Science, pp. 83-103, 2005.

[Tal07a] D. Talia, P. Trunfio; *Toward a synergy between P2P and Grids*, IEEE Internet Computing, Vol. 7, N. 4, pp. 94-96, 2003.

[Tal07b] D. Talia, P. Trunfio, and J. Zeng; *Peer-to-peer models for resource discovery in large-scale grids: A scalable architecture*, In High Performance Computing for Computational Science - VECPAR 2006, pp. 66–78, 2007.

[Tie02] B. Tierney et al.; *A Grid Monitoring Architecture*, Grid Forum, Grid Performance Working Group, 2002.

[Tru07] P. Trunfio et al.; *Peer-to-Peer resource discovery in Grids: Models and systems,* Future Generation Computer Systems, Elsevier, Vol. 23, N. 7, pp. 864-878, 2007.

[Wat05] P. Watson; *Databases in grid applications: Locality and distribution,* British national conference on databases, LNCS, No. 22, Vol. 3567, pp. 1-16, 2005.

[Wie92] G. Wiederhold; *Mediators in the Architecture of Future Information Systems,* Journal of IEEE Computer, Vol. 25, N.3, pp. 38-49, 1992.

[Woh07] A. Wohrer, P. Brezany; *A monitoring Service for Relational Database to Support Advanced Data Integration on the Grid*, 1st Intl. Conf. on Complex, Intelligent and Software Intensive Systems, pp. 27-34, 2007.

[Wol99] R. Wolski et al.; *The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing,* Journal of Future Generation Computing Systems, Elsevier , Vol. 15, N. 5-6, pp. 757-768, 1999.

[Xia05] Q. Xia et al.; *Fully Decentralized DHT based Approach to Grid Service Discovery using Overlay Networks,* Int. Conf. on Computer and Information Technology, pp.1140-1045, 2005.

[Yan03] B. Yang, H. Garcia-Molina; *Designing a super-peer network,* Proc. of the 19th Intl. Conf. on Data Engineering, pp. 49-60, 2003.

[Yan07] M. Yan et al.; *Study of Grid Resource Discovery Based on Mobile Agent,* Proc. of the 3rd Intl. Conf. on Semantics, Knowledge and Grid, pp. 570-571, 2007.

[Yu06] J. Yu et al.; *Grid Resource Management Based on Mobile Agent,* Proc. of the Intl. Conf. on Computational Intelligence for Modeling Control and Automation and Intl. Conf. on Intelligent Agents Web Technologies and Intl Commerce, pp. 255-256, 2006.

[Zan05] S. Zanikolas, R. Sakellariou; *A taxonomy of grid monitoring systems,* Future Generation Computer Systems, Elsevier , Vol. 21, pp. 163-188, 2005.

[Zhu05] H. Zhuge et al, *Query Optimization in Database Grid*, G.C. Fox (Eds.): GCC 2005, LNCS 3795, pp. 486 – 497, 2005

[Zhu06] P-Y. ZHUI et al.; *An agent federations based infrastructure for grid monitoring and discovery*, Proceedings of the Fifth International Conference on Machine Learning and Cybernetics, Dalian, 13-16, 2006.

Abstract: The distributed data management in grid systems raises new problems and presents real challenges: resource discovery, resource allocation, replication, monitoring services for query optimization …etc. Grid systems differ mainly from parallel and distributed systems by the two characteristics: the large scale and the system instability (i.e. the dynamicity of nodes).

In this thesis, we are interested in the resource discovery phase for an efficient query evaluation in data grid environments. First, we present a state of the art on the main research works of resource discovery by focusing on the important criteria (e.g. scaling, reliable discovery, maintenance cost) for data source discovery which is specific to data grid environments. In this perspective, we propose a new method of data source discovery based on Distributed Hash Tables (DHT) allowing a permanent access -in the presence of the dynamicity of nodes- from any node of a Virtual Organization VO_{local} towards all other VO_i ($i \neq local$) in the system with a minimum maintenance cost between the DHT. After the resource discovery phase, it is very important to monitor the current state of resources especially that these last ones are shared on very large scale environments. The resource monitoring can be made during the initial allocation or the execution phase, in order to take decisions on the choice of the execution node of a join (or of a part of a join) for example. In this context, we propose a method considering the variation of host and network parameter values, at runtime, in the calculation of the response time of a relational operation. The proposed method integrates monitoring information into an execution model based on mobile agents developed in the Pyramid team. Finally, we validate our proposals by a performance evaluation.

Keywords: Data Grid, Query Evaluation, Resource Discovery, Resource Monitoring, Performance Evaluation.