RenPar'17 / SympA'2006 / CFSE'5 / JC'2006
Canet en Roussillon, 4 au 6 octobre 2006

Un protocole de découverte de ressources optimisé pour l'allocation de ressources dans les grilles

Emmanuel Jeanvoine
EDF R&D/IRISA
Emmanuel.Jeanvoine@irisa.fr

Christine Morin
IRISA/INRIA
Christine.Morin@irisa.fr

Daniel Leprince
EDF R&D
Daniel.Leprince@edfgdf.fr

Résumé
L'utilisation des grilles de calcul pour la simulation numérique est attractive compte-tenu de la puissance de calcul qu'elles offrent. Ceci dit, les grilles demeurent complexes à exploiter. Pour cela nous proposons le système Vigne destiné à simplifier l'utilisation d'une grille. En particulier, Vigne offre la possibilité d'allouer des ressources à des applications en masquant la distribution et la dynamicité des ressources. Dans ce papier, nous présentons un protocole fondé sur un réseau logique non-structuré et le mécanisme de marches aléatoires permettant d'effectuer de la découverte de ressources. Notre contribution réside dans l'optimisation de ce protocole pour une utilisation dans le cadre de l'allocation de ressources dans les grilles de calcul. Des expérimentations réalisées sur la plate-forme Grid'5000 nous ont permis de vérifier l'efficacité de notre protocole en le comparant à des protocoles existants dans l'état de l'art.

Mots-clés : intergiciel de grille, découverte de ressources, allocation de ressources, marches aléatoires

1. Introduction

La simulation numérique occupe une place importante dans de nombreux domaines scientifiques où l'expérimentation est impossible. Beaucoup d'applications scientifiques requièrent une puissance de calcul bien plus importante que ce que pourrait fournir une simple station de travail. Les grilles de calculateurs sont naturellement devenues très attractives pour ce type d'applications.
Utiliser la puissance de calcul fournie par une grille dont les ressources sont volatiles et dispersées sur des sites géographiquement éloignés est encore aujourd'hui un défi.
Nous proposons le système Vigne dont le but est de simplifier l'utilisation d'une grille. Nous avons conçu ce système en tenant compte de l'aspect très dynamique des ressources qui sont réparties dans un environnement de grande échelle. Pour simplifier l'utilisation d'une grille, Vigne propose une vue de type *Système à Image Unique* (SIU) qui occulte la distribution des ressources. Les utilisateurs ont juste à fournir une description des besoins des applications qu'ils veulent exécuter, comme ils le feraient avec un gestionnaire de travaux. Pour réaliser une exécution transparente des applications, Vigne possède un service d'allocation de ressources dont l'objectif est de découvrir des ressources correspondant aux besoins exprimés par l'utilisateur et de sélectionner la ressource la plus adaptée parmi celles qui ont été découvertes. L'étape de découverte est réalisée par un service possédant de bonnes propriétés de passage à l'échelle et fournissant la possibilité d'effectuer des recherches multi-critères. Pour concevoir le service de découverte de ressources, nous avons utilisé des mécanismes fondés sur des réseaux logiques non structurés et des protocoles de marches aléatoires.

Notre contribution est la conception et l'implémentation dans le système Vigne d'un protocole de découverte de ressources qui minimise la bande passante utilisée et qui maximise la qualité des résultats obtenus après une étape de découverte de ressources. La qualité des ressources découvertes est évaluée en fonction de leur utilité pour l'allocation de ressources, i.e. les ressources non chargées sont plus utiles que les ressources récemment allouées. Notre approche consiste à étendre le protocole de marche aléatoire classique avec un mécanisme de dissémination d'informations et un cache géré avec une politique favorisant la qualité des résultats obtenus. Nous avons évalué notre protocole sur la plate-forme Grid'5000 afin d'en montrer l'efficacité en le comparant au protocole basique de marche aléatoire et à un protocole existant dans l'état de l'art.
L'article est organisé de la façon suivante. Nous présentons dans la section 2 la problématique de la découverte de ressources dans les grilles ainsi que les travaux de l'état de l'art. Dans la section 3, nous présentons un protocole classique de découverte de ressources et nous proposons des extensions à y apporter afin d'accroître son efficacité dans le contexte d'utilisation de l'allocation de ressources dans une grille. Dans la section 4 nous présentons une évaluation du protocole que nous proposons. Finalement, nous concluons et présentons les travaux futurs dans la section 5.

2. La découverte de ressources dans les grilles

2.1. Objectifs d'un système de grille pour la découverte de ressources

Afin d'exécuter les applications des utilisateurs, un système de grille doit allouer des ressources aux applications. La figure 1 montre les étapes de l'allocation de ressources dans un système de grille. Étant donné l'hétérogénéité des ressources d'une grille en termes de matériel ou de logiciel, un utilisateur fournit une description des ressources requises pour l'application qu'il souhaite exécuter. La figure 2 montre un exemple de description des besoins d'une applications, comme celles utilisés dans le système Vigne. Nous pouvons voir que cette description est composée de plusieurs attributs et que certains attributs comme `OS_Version` ou `Mem` n'ont pas une valeur scalaire, mais une valeur définie par un intervalle de valeurs. Ainsi, le système doit être capable de comprendre une contrainte telle que : *J'ai besoin d'un noyau Linux dont la version est au moins 2.6.12 pour exécuter mon application*. A partir des contraintes pour l'exécution de l'application, le service de découverte de ressources doit trouver des ressources qui peuvent les satisfaire. Une fois que des ressources ont été trouvées, le service de sélection de ressources doit choisir la plus adaptée en fonction d'une politique donnée, par exemple en choisissant la ressource la moins chargée.

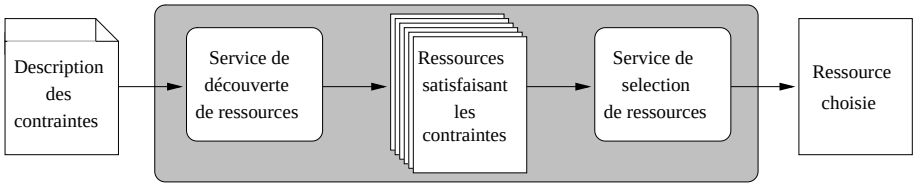

FIG. 1 – Service d'allocation de ressources

```
<ApplicationRequirements>
  <Arch>X86_64</Arch>
  <OS>LINUX</OS>
  <OS_Version>2.6.12+</OS_Version>
  <Net>Infiniband</Net>
  <Mem>2048+</Mem>
  <Nodes>6</Nodes>
  <Libs>
    <Mpi>LAM_MPI</Mpi>
  </Libs>
</ApplicationRequirements>
```

FIG. 2 – Exemple d'une description d'application

2.2. État de l'art

La découverte de ressources est une question qui est souvent étudiée dans les systèmes de grilles. Dans cette section, nous présentons plusieurs approches de l'état de l'art.

Les approches centralisées comme XtremWeb [3] fournissent un mécanisme de découverte de ressources très efficace qui est fondé sur une entité centrale connaissant l'intégralité des ressources du système. XtremWeb est fondé sur un modèle client/travailleur/coordinateur. Lorsqu'un travailleur rejoint le système, il s'enregistre auprès du coordinateur. Lorsqu'un client soumet une tâche, il contacte le coordinateur. Ainsi, le coordinateur choisit un travailleur libre pour l'exécution de la tâche. Les problèmes avec ce type d'approche sont que le coordinateur est supposé être sur une ressource fiable et qu'il peut devenir goulot d'étranglement s'il est trop sollicité. Étant donné que nous ne souhaitons pas considérer de ressources stables pour un coordinateur et que nous considérons un système à grande échelle, une approche centralisée n'est pas appropriée pour nos travaux.

Les approches comme Globus [4] ou Active Yellow Pages [14] (ActYP) possèdent un mécanisme de découverte de ressources fondé sur des base de données. Ces mécanismes ne sont pas complètement centralisés puisque les bases de données peuvent être distribuées. La boîte à outils Globus [4] utilise le *Monitoring and Discovery System* (MDS) fondé sur des documents XML et XPath pour sélectionner des ressources dans les documents XML. Les base de données du MDS sont organisées hiérarchiquement pour supporter le passage à l'échelle. Un problème avec cette approche hiérarchique est que les administrateurs d'un MDS doivent connecter leur MDS à un MDS de plus haut niveau. De plus, si un MDS connaît une défaillance, tout le sous-arbre associé à ce MDS est perdu. ActYP [14] utilise un ensemble de bases de données pour stocker la description des ressources qui composent le système et offre un mécanisme pour agréger des requêtes similaires afin d'optimiser le temps de réponse. Minimiser la latence du mécanisme de découverte de ressource peut-être très intéressant pour plusieurs classes d'application telles que les simulations paramétriques. La tolérance aux défaillances est supportée par ActYP via une duplication des bases de données. Le principal problème avec Globus et ActYP réside dans le fait que l'enregistrement des ressources n'est pas automatique et doit être effectué par un administrateur. Comme nous souhaitons fournir des propriétés d'auto-organisation, de telles approches ne peuvent pas être utilisées.

D'autres approches distribuées sont fondées sur les technologies issues des travaux sur les systèmes pair-à-pair. Deux types de travaux existent, ceux fondés sur les réseaux logiques structurés et ceux fondés sur les réseaux logiques non-structurés. Les deux approches furent initialement développées dans le contexte du partage de fichiers. Les réseaux logiques structurés

comme Pastry [13]fournissent des mécanismes pour organiser les nœuds en fonction d'un nom numérique. Ces réseaux structurés sont adaptés pour effectuer des recherches d'éléments dont le nom numérique est connu. Ceci dit, si juste une description de l'élément fondée sur des critères est connue et que le nom numérique de cet élément est inconnu, la recherche reste possible mais pose un problème d'équilibrage de charge. Pour résoudre ce problème, des projets comme [1], [7] ou [8] ont pour but de fournir la possibilité d'effectuer des recherches multi-critères avec des critères pouvant être définis sur un intervalle de valeurs dans un réseau logique structuré. Toutefois, ces projets nécessitent d'utiliser un réseau logique dédié et impliquent un surcoût élevé de consommation de bande passante pour maintenir le réseau dans un état cohérent. Les réseaux logiques non-structurés comme Gnutella [6] ne fournissent pas de modèle d'organisation spécifique pour les nœuds comme celui des réseaux structurés. Dans de tels réseaux logiques, les nœuds sont connectés à un nombre limité de voisins. Avec Scamp [5], ce nombre est $\log(n)$ en moyenne, où n est le nombre de nœuds dans le réseau. Avec les réseaux logiques non-structurés, il est possible d'effectuer des recherches multi-critères en utilisant un mécanisme d'inondation. L'inondation basique ne possède pas de bonnes propriétés de passage à l'échelle à cause de la bande passante utilisée. Toutefois des optimisations comme les mécanismes de marches aléatoires peuvent être utilisés pour effectuer des recherches dans des systèmes à grande échelle. Dans [9], les auteurs étudient la convergence entre les communautés pair-à-pair et grille. Ils ont évalué des protocoles fondés sur les marches aléatoires dans un simulateur. Ce travail est relativement proche du nôtre pour ce qui est des techniques utilisées étant donné que nous avons choisis de fonder le service de découverte de ressources de Vigne sur un réseau logique non structuré pour la possibilité qu'il offre d'effectuer des recherches multi-critères et sur les marches aléatoires pour leur légèreté. Ceci dit, le contexte d'application est différent puisque les auteurs de [9] ont évalué la métrique du nombre de sauts pour une requête dans le cadre de partage de fichiers alors que nous mesurons l'intérêt des ressources découvertes en vue d'effectuer une allocation de ressources. Notre contribution est l'étude et l'amélioration des ces protocoles en les appliquant à la découverte de ressources dans un environnement de grille et avec comme objectif de réaliser une allocation de ressources à partir des ressources découvertes.

3. Étendre les mécanismes de marches aléatoires pour découvrir des résultats de haute qualité

3.1. Modèle du système

Plusieurs propriétés sont fortement souhaitées pour le mécanisme de découverte de ressources d'un système de grille. Tout d'abord, ce mécanisme doit être capable de trouver des ressources efficacement, sans consommer trop de bande passante. Ensuite, ce mécanisme doit pouvoir passer à l'échelle, typiquement il doit fonctionner dans une grille composée de plusieurs milliers de ressources. Dans une telle grille, les ressources ne peuvent pas être considérées comme stables. Elles sont en effet amenées à quitter le système pour cause de défaillances ou départs volontaires. Malgré ce comportement très dynamique, le mécanisme de découverte de ressources doit garder une efficacité constante. Finalement, ce mécanisme doit fournir des résultats de *haute qualité*. Par résultats de *haute qualité*, nous entendons résultats utiles pour l'allocation de ressources et plus particulièrement pour la sélection des ressources. En effet, si le mécanisme de découverte de ressources retourne systématiquement le même ensemble de ressources pouvant convenir à une exécution, le service de sélection devra effectuer sa sélection

parmi ces mêmes ressources. Cela impliquera que ces ressources se retrouveront surchargées.

3.2. Principes de conception

Compte-tenu des suppositions faites dans la section 3.1, une approche distribuée fondée sur les réseaux logiques est adaptée pour la grande échelle et la dynamicité des ressources. En effet, ces propriétés sont intrinsèques aux réseaux logiques.
Étant donné que les requêtes de découverte de ressources devant être réalisées sont multi-critères, les réseaux logiques non structurés sont plus adaptés que les réseaux structurés. Ainsi, nous choisissons de fonder le service de découverte de ressources de Vigne sur un réseau logique non structuré.
Pour éviter le gaspillage de bande passante avec l'inondation de requêtes, nous choisissons d'utiliser les mécanismes de marches aléatoires pour la propagation des requêtes. Ainsi, la propagation des requêtes est suspendue dès qu'un nombre suffisant de ressources a été découvert, alors qu'avec une inondation classique cela est impossible.

3.2.1. Le protocole de marches aléatoires

Le principe de fonctionnement d'un protocole de marches aléatoires est le suivant. Lorsqu'un nœud reçoit une requête, il la propage à seulement un de ses voisins aléatoirement choisis, ceci contrairement à l'inondation classique où la requête est propagée à tous les voisins. Comme avec l'inondation, une requête est propagée de nœuds en nœuds jusqu'à une profondeur donnée. Pour initialiser le protocole, le nombre de résultats souhaité à l'issue des marches aléatoires doit être donné. Si après une marche aléatoire, les résultats obtenus sont inférieurs aux résultats requis, une autre marche aléatoire est effectuée, et ceci tant qu'une temporisation n'a pas expirée. Le principal avantage de ce protocole par rapport à l'inondation est sa légèreté puisque le nombre de messages propagés est fortement réduit. De plus, le risque de goulot d'étranglement est limité puisque l'initiateur d'une requête peut recevoir au maximum autant de réponses à la fois que la profondeur de la marche aléatoire. Dans le cas de l'inondation, ce nombre de réponses est exponentiel.

3.2.2. Le protocole de marches aléatoires avec cache

Dans [9], les auteurs proposent une optimisation du protocole de marches aléatoires en utilisant un cache. L'objectif en utilisant un cache est de minimiser le coût en bande passante des requêtes de découverte de ressources. Ainsi, lorsqu'une découverte de ressources est réalisée et que des résultats sont obtenus, l'adresse des ressources découvertes est mise en cache. Quand une requête est soumise et qu'un nombre de résultats est demandé, le système vérifie si le cache possède des résultats pouvant être utilisés et il complète si besoin en effectuant d'autres marches aléatoires pour obtenir le nombre de résultats demandés. Les auteurs de [9] ont étudié trois stratégies de gestion de cache qui sont : aléatoire, plus grand nombre de réponses et apprentissage. Par gestion de cache nous entendons trois actions qui sont : le choix des données à insérer dans le cache, le choix des données à supprimer si le cache est plein et la sélection de données dans le cache lorsque celui-ci offre plus d'informations que ce qui est requis. La stratégie aléatoire consiste à ne favoriser aucun nœud et à effectuer les actions ci-dessus aléatoirement. Pour les deux autres stratégies, le cache mémorise en plus de l'adresse des ressources, le nombre de réponses positives ou négatives qu'elles ont effectué. La stratégie du plus grand nombre de réponses consiste à favoriser les nœuds qui ont répondu (positivement ou négativement) au plus grand nombre de requêtes). La stratégie d'apprentissage consiste à favoriser les nœuds qui ont le mieux répondu aux précédentes requêtes, en ne prenant donc en compte que les réponses positives.

3.2.3. Optimisations du protocole de marches aléatoires pour des résultats de haute qualité

Nous proposons d'optimiser le protocole présenté dans la section 3.2.2 pour obtenir un nouveau protocole favorisant les résultats de *haute qualité*. Nous appelons ce protocole : *Random Walk protocol Optimized for Grid Scheduling* (RW-OGS).

En utilisant un cache, le nombre de messages propagés peut être réduit, ainsi que la durée d'une découverte de ressources. Toutefois, utiliser un cache peut poser le problème de l'appauvrissement des données stockées.

L'utilisation d'un cache avec une stratégie telle que celles présentées dans la section 3.2.2 induit un effet non souhaité. En effet, une fois que le cache contient au moins autant d'informations que de résultats souhaités à la suite d'une découverte de ressources, aucune nouvelle marche aléatoire ne sera réalisée et les résultats obtenus seront systématiquement les mêmes. Ces résultats n'auront donc pas la *haute qualité* discutée dans la section 3.1 puisque le choix par le service de sélection se fera parmi ces mêmes ressources qui deviendront au final surchargées.

Tout d'abord, donner une date de péremption aux données du cache peut réduire cet appauvrissement, mais un compromis doit être trouvé pour conserver des données *fraîches* sans en supprimer trop souvent et donc perdre l'intérêt du cache.

Ensuite, nous utilisons une autre stratégie de gestion de cache pour augmenter la qualité des résultats en vue d'effectuer des allocations de ressources. L'idée est la suivante : quand une ressource est choisie par le service de sélection de ressource (cf. figure 1), la date de cette sélection est enregistrée dans les méta-données de la ressource sélectionnée dans le cache. Ensuite, les données choisies sont celles qui ont la dernière date de sélection la plus ancienne. Ainsi, la probabilité pour le service de découverte de ressources de retourner un ensemble de ressources de *haute qualité* est plus élevée qu'avec les stratégies énoncées dans la section 3.2.2. Par conséquent les ressources récemment allouées seront moins facilement réutilisées et donc moins facilement surchargées.

Pour accélérer le remplissage du cache et le rafraîchissement des informations mises en cache, nous ajoutons un mécanisme de dissémination des informations découvertes vers certains nœuds. Ces nœuds sont les voisins dans le réseau logique non structuré du nœud initiateur de la découverte de ressources. De plus, les informations de la ressource choisie par l'allocateur de ressources ne sont pas disséminées aux voisins. En effet, avoir une ressource récemment allouée dans son cache ne contribue pas à avoir des résultats de *haute qualité* après une découverte de ressources.

```
struct nodeInformation {
  nodeAddress;
  nodeDescription;
  cacheArrivalDate;
  lastAllocationDate;
}
```

FIG. 3 – Structure des données mises en cache

Utiliser le protocole *RW-OGS* n'induit qu'un très faible surcoût de consommation mémoire pour ce qui concerne les méta-données du cache. La figure 3 montre les informations stockées dans le cache pour un nœud. Les attributs `nodeAddress` et `nodeDescription` sont requis par n'importe quelle stratégie de gestion de cache. Nous avons juste ajouté les attributs

`cacheArrivalDate` et `lastAllocationDate` pour supprimer les informations les plus anciennes et pour ajouter une gestion de cache qui favorise les informations pertinentes pour l'allocation de ressources. Dans notre implémentation, la taille de la structure `nodeInformation` est d'environ 300 octets.

3.2.4. Algorithme du protocole RW-OGS

Le protocole *RW-OGS* fonctionne ainsi. Lorsqu'un nœud initie une allocation de ressources (cf. algorithme 1) à la suite d'une soumission de tâche par un utilisateur, il effectue une découverte de ressources, ensuite il choisit une ressource parmi celles découvertes et finalement il dissémine les informations récupérées. Pour effectuer une découverte de ressources (cf. algorithme 2, un nœud examine le contenu de son cache afin de trouver des ressources satisfaisant les contraintes pour l'exécution de la tâche. S'il trouve des ressources satisfaisant les contraintes, il leur demande directement leur charge. Si le cache contient plus de ressources satisfaisant les contraintes que de résultats souhaités, les ressources les plus récemment allouées ne sont pas choisies. Lorsqu'un nœud reçoit une requête de découverte de ressources(cf. al-

Algorithm 1 Fonction PerformResourceAllocation()

```
1: function PERFORMRESOURCEALLOCATION(requirements)
2:     responsesSet = OptimizedRandomWalk(WANTED_RESULTS, requirements)
3:     selectedResource = SelectResource(responsesSet)
4:     DisseminateNodeInformation(nodeInformation, selectedResource)
5: end function
```

Algorithm 2 Fonction OptimizedRandomWalk()

```
1: function OPTIMIZEDRANDOMWALK(Nb_results, requirements)
2:     responsesSet ← ∅
3:     suitableCachedNodes ← GetNodeInformationFromLocalCache(requirements)
4:     for each node ∈ suitableCachedNodes do
5:         AskNodeInformation(node, requirements)
6:     end for each
7:     if |suitableCachedNodes| < Nb_results then
8:         RandomWalk(Nb_results − |suitableCachedNodes|, requirements)
9:     end if
10:    while (|responsesSet| < Nb_results) ∧ ¬(EndOfDiscovery()) do
11:        nodeInformation ← WaitForResponse()
12:        responsesSet ← responsesSet ∪ {nodeInformation}
13:        if ¬(nodeInformation ∈ localCache) then
14:            localCache ← localCache ∪ {nodeInformation}
15:        end if
16:    end while
17:    return responsesSet
18: end function
```

gorithme 3), il compare les contraintes de l'application avec les ressources qu'il peut offrir et répond à l'initiateur s'il peut satisfaire les contraintes. La réponse contient les caractéristiques réelles de la ressource ainsi que sa charge. Si son cache ne contient pas suffisamment de res-

Algorithm 3 Function ReceiveResourceDiscoveryQuery()

```
1: procedure RECEIVERESOURCEDISCOVERYQUERY(requirements, sender)
2:     if (CompareResourceInfo(requirements, localResources) then
3:         SendResponse(sender, localResources, load)
4:     end if
5: end procedure
```

sources pouvant satisfaire la requête, l'initiateur de la découverte de ressources effectue des marches aléatoires pour compléter les résultats. Lorsque l'initiateur reçoit une réponse d'une ressource, il ajoute dans le cache les informations relatives à cette ressource (cf. figure 3) si les informations n'y sont pas déjà présentes.
Lorsque suffisamment de réponses sont obtenues, ou après un temps donné, le service de sélection de ressources choisit une ressource parmi celles découvertes et les informations concernant les ressources découvertes, sauf celle choisie, sont disséminées dans le voisinage de l'initiateur(cf. algorithme 4). Quant à la ressource choisie, un message est envoyé pour informer les voisins de l'initiateur de l'allocation, et ainsi pour qu'ils mettent à jour l'information d'allocation dans leur cache s'ils connaissaient déjà la ressource allouée. Lorsqu'un nœud reçoit une

Algorithm 4 Fonction DisseminateNodeInformation()

```
1: procedure DISSEMINATENODEINFORMATION(nodeInfo, selectedResource, responsesSet)
2:     for each response ∈ responsesSet do
3:         if response ≠ selectedResource then
4:             for each node ∈ structuredNetworkNeighborood do
5:                 SendNodeInformation(nodeInfo, node)
6:             end for each
7:         else
8:             for each node ∈ structuredNetworkNeighborood do
9:                 AskToTagAsAllocated(nodeInfo, node)
10:            end for each
11:        end if
12:    end for each
13: end procedure
```

information qui a été disséminée (cf. algorithme 5), il met à jour son cache. Si le cache est plein, le nœud supprime les informations relatives à la ressources la plus récemment allouée.

4. Évaluation

Le protocole *RW-OGS* a été mis en œuvre dans le système Vigne en vue de son évaluation sur la plate-forme Grid'5000.

Algorithm 5 Fonction ReceiveNodeInformation()

```
1: procedure RECEIVENODEINFORMATION(nodeInfo)
2:     n ← GetNodeInformationFromLocalCache(nodeInfo)
3:     if (n = ∅) ∨ ¬(CompareResourceInfo(n, nodeInfo)) then
4:         if |localCache| ≥ CacheSize then
5:             RemoveMostRecenltyAllocatedCachedNode()
6:         end if
7:         localCache ← localCache ∪ nodeInformation
8:     end if
9: end procedure
```

4.1. Implémentation dans le système Vigne

Vigne est un système conçu pour simplifier l'utilisation d'une grille de grande échelle constituée de plusieurs milliers de ressources ayant un comportement dynamique en offrant une vue de type SIU qui masque la distribution des ressources. Vigne peut fédérer des ressources hétérogènes exploitées en mode interactif ou en mode différé. De plus, Vigne possède de propriétés d'auto-organisation et d'auto-réparation qui permettent aux utilisateurs et administrateurs de ne pas se préoccuper de la dynamicité des ressources. Nous présentons dans [10] l'architecture générale de Vigne. L'implémentation courante de Vigne inclut divers services comme : un service de partage de mémoire transparent [12], un service d'allocation de ressources, des interfaces génériques avec différents gestionnaires de ressources (Linux, Kerrighed, OpenPBS) ou encore des interfaces pour l'utilisateur comme une interface de soumission de tâche par exemple.

Afin d'obtenir un système ayant de bonne propriétés de passage à l'échelle, d'auto-organisation et d'auto-réparation, les services de Vigne sont fondés sur des réseaux logiques [11]. Pour donner quelques détails, Vigne possède deux services de communication. Le premier permet d'effectuer des communications au-dessus d'un réseau logique structuré. L'implémentation de ce réseau logique est fondée sur Pastry [13]. Le second service de communication permet de réaliser des communication au-dessus d'un réseau logique non structuré. L'implémentation de ce service est fondée sur Scamp [5] pour la constitution du réseau. Dans [11] nous expliquons que l'utilisation conjointe des deux réseaux logiques n'induit qu'un faible surcoût. En effet nous avons réalisé une implémentation allégée du réseau logique non structuré en utilisant certaines propriétés du réseau logique structuré pour l'amorçage, l'espace de désignation, la détection de défaillances et pour éviter les partitions.

L'allocation de ressources étant l'un des objectifs de Vigne, nous avons mis en œuvre ce service. Il est divisé en deux parties comme le montre la figure 1. Notre implémentation du service d'allocation de ressources permet d'implémenter divers protocoles de découverte de ressources fondés sur un réseau logique non structuré. Actuellement, nus avons implémenté trois protocoles qui sont l'inondation, la marche aléatoire et la marche aléatoire avec cache. Nous avons aussi implémenté deux stratégies de gestion de cache qui sont l'apprentissage (cf. section 3.2.2) et *RW-OGS* (cf. section 3.2.3).

4.2. Expérimentations

4.2.1. Objectifs

Nous avons évalué l'efficacité du protocole *RW-OGS* décrit dans la section 3.2.3 en le comparant au protocole classique de marches aléatoires (cf. section 3.2.1) et au protocole de marches aléa-

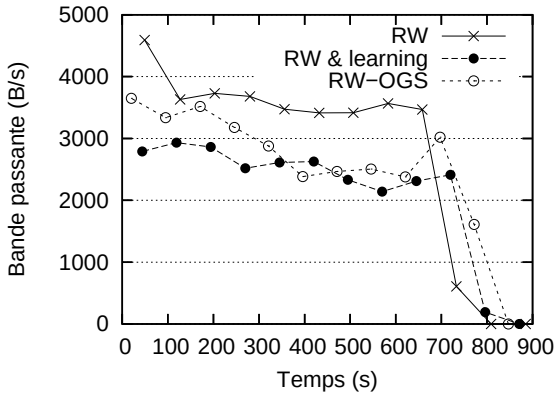

FIG. 4 – Comparaison de la bande passante consommée par les trois protocoles

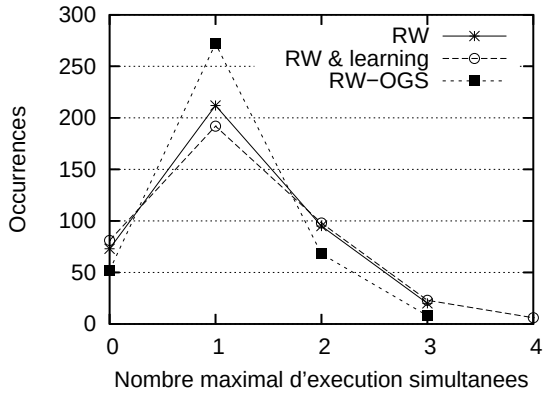

FIG. 5 – Occurrences du maximum d'exécutions concurrentes par nœud

toires étendu avec un cache possédant une stratégie d'apprentissage (cf. section 3.2.2). Ces trois protocoles ont été comparés en fonction de deux critères qui sont la bande passante consommée et la qualité des résultats obtenus après une étape de découverte de ressources.
La qualité des résultats obtenus est évaluée par la capacité du protocole à ne pas découvrir systématiquement les mêmes ressources, en particulier les ressources qui auraient été allouées récemment.

4.2.2. Conditions expérimentales

Pour réaliser ces expérimentations, nous avons déployé le système Vigne sur la plate-forme nationale d'expérimentation Grid'5000 [2]. Nous avons utilisés 100 machines de la grappe rennaise sur laquelle nous avons lancé 400 nœuds du système Vigne.
Pour évaluer les deux critères présentés dans la section 4.2.1, nous avons procédé de la façon suivante. Tout d'abord, nous avons divisé les ressources utilisées en quatre types. Ensuite nous avons lancé quatre applications, chacune nécessitant un des quatre types de ressources et chacune composée de 200 processus durant 180 secondes. Le lancement de chaque processus a été espacé d'une seconde. Ce type d'application représente les applications paramétriques, grandes consommatrices de puissance de calcul. Nous avons mesuré la bande passante consommée par les trois protocoles grâce à une fonctionnalité de Vigne qui permet de mesurer la bande passante consommée par chaque service, en particulier celui de découverte de ressources. La qualité des résultats obtenus avec le service de découverte de ressources est évaluée indirectement par la mesure de l'efficacité de l'allocateur de ressources. Étant donné que la politique de sélection de ressources est de choisir la ressource la moins chargée parmi celles découvertes, plus la qualité des résultats obtenus est élevée, plus l'allocateur de ressources aura de chance d'allouer une ressource peu chargée. Donc nous avons mesuré le nombre maximal de processus s'exécutant en concurrence par nœud. Plus ce nombre est faible, plus le mécanisme de découverte de ressources fournit des résultats efficaces.
Pour ces expérimentations, nous avons fixé la taille des caches à 50, la profondeur d'une marche aléatoire à 6 et la durée maximale d'une découverte de ressources à 60 secondes.

4.2.3. Résultats

La figure 4 montre la bande passante consommée par les protocoles : marche aléatoire (*RW*), marche aléatoire avec cache et gestion du cache par apprentissage (*RW & learning*) et marche

aléatoire optimisée pour l'allocation de ressources (*RW-OGS*). Nous pouvons observer qu'en régime permanent, le protocole *RW* est celui qui consomme le plus de bande passante et que *RW-OGS* consomme légèrement plus de bande passante que *RW & learning*. Le protocole *RW* est en effet celui qui génère le plus de messages puisqu'à chaque requête, il doit effectuer plusieurs marches aléatoires. La faible consommation du protocole *RW & learning* est due au fait que chaque nœud n'effectue qu'une seule fois une découverte de ressources. En effet, une fois que les résultats d'une découverte de ressources sont en cache, il ne reste qu'à demander directement leur charge aux nœuds du cache sans avoir à refaire de nouvelles marches aléatoires. C'est donc le protocole qui permet de minimiser au maximum la bande passante consommée. Le protocole *RW-OGS* obtient quasiment le même résultat que *RW & learning*, malgré les messages d'information d'allocation qui sont envoyés. Cela est en fait compensé par la dissémination des informations vers les voisins lorsqu'une découverte de ressources est effectuée qui permet d'éviter une partie des marches aléatoires devant être réalisées lorsque un nœud effectue pour la première fois une découverte de ressources.
La figure 5 montre la répartition des exécutions de processus qui ont eu lieu de façon concurrente durant l'expérience. Un protocole parfait aurait dû donner une répartition où il y a un maximum de nœuds qui exécutent un seul processus à la fois. Nous pouvons remarquer que le protocole *RW-OGS* donne le meilleur résultat puisqu'il compte plus d'occurrences où il n'y a pas eu de concurrence d'exécution (environ 275 occurrences), c'est à dire où il n'y avait au maximum qu'un processus s'exécutant à la fois, que les autre protocoles (environ 200 occurrences). De plus, c'est celui qui compte le moins d'occurrences pour une concurrence de 2, 3 et 4 processus par nœud, ce qui est un avantage car la concurrence d'exécution réduit les performances pour l'exécution de chaque processus et donc augmente son temps d'exécution. C'est aussi celui qui occupe le plus de ressources puisque seulement 50 nœuds n'ont pas été utilisés alors que pour les protocoles *RW* et *RW & learning*, environ 75 nœuds n'ont pas été utilisés.
De ces résultats, nous pouvons estimer que le protocole *RW-OGS* offre de meilleurs performances dans un contexte d'allocation de ressources que les autre protocoles, tout en ayant une consommation de bande passante proche d'une consommation optimale.

5. Conclusion et travaux futurs

Ce papier décrit un protocole de découverte de ressources conçu pour l'allocation de ressources dans les grilles de grande taille. Ce protocole est fondé sur un réseau logique non structuré et des optimisations du protocole classique de marche aléatoire.
Nous avons évalué ce protocole sur la plate-forme Grid'5000 en le comparant au protocole classique de marche aléatoire et à un autre protocole de l'état de l'art utilisant un cache. Les résultats montrent que ce protocole est le plus efficace pour ce qui concerne sont intérêt dans un service d'allocation de ressources et que sa consommation en bande passante demeure proche d'une consommation optimale.
Des travaux futurs seront dédiés à l'évaluation des protocoles présentés en faisant varier divers paramètres comme la taille du cache, le nombre de résultats demandés après une découverte de ressources ou encore la profondeur d'une marche aléatoire. Il serait aussi intéressant d'évaluer ces protocoles en environnement dynamique. De nouveaux protocoles de découverte de ressources pourront aussi être implémentés dans le système Vigne. En particulier, il est possible d'implémenter des protocoles de découverte de ressources fondés sur un réseau logique structuré en utilisant des mécanismes de publication/souscription. Vigne possède seulement une politique de sélection de ressources utilisant la charge des ressources comme critère de choix. Il pourrait être intéressant de créer de nouvelles politiques et de mesurer les interactions avec

les protocoles de découverte de ressources.

6. Remerciements

Nous tenons à remercier Louis Rilling pour ses travaux sur la conception et la mise en œuvre de plusieurs services de Vigne sur lesquels reposent ces travaux.

Bibliographie

1. Bharambe (A. R.), Agrawal (M.) et Seshan (S.). – Mercury : supporting scalable multi-attribute range queries. *In : Proc. of SIGCOMM 2004*. pp. 353–366. – New York, NY, USA, 2004.
2. Cappello (F.), Desprez (F.), Dayde (M.), Jeannot (E.), Jegou (Y.), Lanteri (S.), Melab (N.), Namyst (R.), Primet (P.), Richard (O.), Caron (E.), Leduc (J.) et Mornet (G.). – Grid'5000 : A large scale, reconfigurable, controlable and monitorable grid platform. *In : Grid2005 6th IEEE/ACM International Workshop on Grid Computing*. – 2005.
3. Cappello (F.), Djilali (S.), Fedak (G.), Herault (T.), Magniette (F.), Néri (V.) et Lodygensky (O.). – Computing on large-scale distributed systems : XtremWeb architecture, programming models, security, tests and convergence with grid. *Future Generation Computer Systems*, vol. 21, n3, mars 2005, pp. 417–437.
4. Foster (I.) et Kesselman (C.). – Globus : A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, vol. 11, n2, Summer 1997, pp. 115–128.
5. Ganesh (A. J.), Kermarrec (A. M.) et Massoulié (L.). – Peer-to-peer membership management for gossip-based protocols. *IEEE Transactions on Computers*, vol. 52, n2, février 2003.
6. Gnutella. – Web Page : http ://rfc-gnutella.sourceforge.net.
7. Hauswirth (M.) et Schmidt (R.). – An overlay network for resource discovery in grids. *In : Proc. of GLOBE 2005*. – Copenhagen, Denmark, août 2005.
8. Heine (F.), Hovestadt (M.) et Kao (O.). – Towards ontology-driven p2p grid resource discovery. *In : 5th IEEE/ACM International Workshop on Grid Computing*, pp. 76 – 83. – 2004.
9. Iamnitchi (A.), Foster (I.) et Nurmi (D.). – A peer-to-peer approach to resource location in grid environments. *In : Proc. of HPDC 2002*, p. 419. – août 2002.
10. Jeanvoine (E.), Rilling (L.), Morin (C.) et Leprince (D.). – Architecture distribuée pour la gestion des ressources dans des grilles à grande échelle. *In : Actes de NOTERE 2006*. – Toulouse, France, June 2006. To appear.
11. Jeanvoine (E.), Rilling (L.), Morin (C.) et Leprince (D.). – Using overlay networks to build operating system services for large scale grids. *In : Proc. of the 5th International Symposium on Parallel and Distributed Computing (ISPDC 2006)*. – Timisoara, Romania, 6 July 2006. To appear.
12. Rilling (L.) et Morin (C.). – A practical transparent data sharing service for the grid. *In : Proc. Fifth International Workshop on Distributed Shared Memory (DSM 2005)*. – Cardiff, UK, mai 2005.
13. Rowstron (A. I. T.) et Druschel (P.). – Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. *In : Proc. of Middleware 2001*, pp. 329–350. – nov 2001.
14. Royo (D.), Kapadia (N. H.), Fortes (J. A. B.) et Diaz de Cerio (L.). – Active yellow pages : A pipelined resource management architecture for wide-area network computing. *In : Proc. of HPDC 2001*, pp. 147–157. – Washington, DC, USA, 2001.