CARI'2012

Services pour les grilles pair-à-pair

Gueye Bassirou* — Flauzac Olivier** — Niang Ibrahima*

* Département de Mathématiques et d'Informatique
Université Cheikh Anta Diop de Dakar
SENEGAL
bassirou.gueye@ucad.edu.sn, iniang@ucad.sn

** CReSTIC
Université de Reims Champagne-Ardenne
FRANCE
olivier.flauzac@univ-reims.fr

RÉSUMÉ. Les grilles basées sur des architectures pair-à-pair ont été utilisées soit dans le cadre du stockage et du partage de données ; soit dans le cadre de la gestion de calculs. En ce qui concerne la mise en place de services, et plus particulièrement de grilles de services, les solutions proposées sont basées sur des grilles hiérarchiques qui présentent un fort degrè de centralisation, tant en ce qui concerne le déploiement des services que la recherche et l'invocation.
Dans ce papier, nous proposons la spécification d'une "grille pair-à-pair de services auto-gérés". L'objectif est de concevoir une solution auto-adaptative permettant le déploiement et l'invocation de services tout en respectant le paradigme des réseaux pair-à-pair. Le déploiement, comme l'invocation sont totalement délégués à la plate-forme et se feront de manière transparente pour l'utilisateur. La spécification se voudra la plus générique possible, non liée à un réseau pair-à-pair particulier ou à un protocole de gestion de services défini à l'avance. Une fois la spécification présentée, nous proposons différentes solutions d'implémentation des grilles pair-à-pair de services, dont l'objectif sera de montrer comment associer des environnements pair-à-pair et ceux d'exécution de services.

ABSTRACT. The grids based on the peer-to-peer architectures were used whether in the storage and sharing of data whether as part of the management calculations. With regard to the setting up services, especially grids services, the proposed solutions are based on hierarchical grids which present a high degree of centralization, both in regard to the deployment of services that the research and the invocation.
In this paper, we propose the specification of a "grid peer-to-peer of services self-managed". The aim is to design a self-adaptive solution for the deployment and invocation of services while respecting the paradigm of services peer-to-peer. The deployment, as the invocation are completely delegated to the platform and will make of transparent manner to the user. The specification will want the most generic possible not related to a service peer-to-peer or a particular management protocol services defined in advance. Once the presented specification, we offer different solutions implementation of grids peer-to-peer of services, whose purpose will to show how to combine environments peer-to-peer and service execution environments.

MOTS-CLÉS : Réseaux pair-à-pair, Grilles de calcul, Services web

KEYWORDS : Peer to peer networks, Grid computing, Web services.

1. Introduction

Dès leur création, les grilles de calcul[7, 6, 8, 14] ont été conues pour stocker des données ou effectuer des calculs scientifiques. Une nouvelle génération de grilles est apparue par la suite ; les grilles de services, permettant, à l'instar de ce qui est offert sur le WEB avec les services web[12], de mettre en place l'accès à des traitements, tout en exploitant le paradigme de la grille. Ces grilles de services ont permis de distribuer les traitements, pour obtenir une utilisation optimale des ressources, et notamment un équilibrage de la charge des machines. Un effort particulier de normalisation est actuellement fait, afin d'apporter, comme dans le cas des services web, toutes les ressources et les moyens nécessaires au développement d'applications dans ces grilles de services[9, 10].

Les grilles qui exploitent la notion de services sont basées sur des architectures hiérarchiques fortement centralisées [7, 14]. Cette centralisation implique une gestion unifiée des ressources, mais aussi des difficultés à réagir vis à vis des pannes et des fautes qui impactent la communauté.

Dans cet article, nous présentons une spécification originale de gestion des services dans un environnement de grilles de calcul basé sur une architecture pair-à-pair. Le déploiement, comme l'invocation sont totalement délégués à la plate-forme et se feront de manière transparente pour l'utilisateur.
Cet article est organisé ainsi, dans un premier temps, nous allons présenter l'originalité de notre solution, puis étudier les travaux connexes. Ensuite, nous présenterons le modèle que nous utilisons, ainsi que les différents aspects de notre spécification. Par la suite, nous proposerons quelques solutions d'implémentations. Nous terminerons cet article par une conclusion, ainsi que des perspectives de mise en œuvre et d'études.

2. Contribution

La spécification que nous proposons présente l'originalité de ne pas lier l'infrastructure pair-à-pair à la plate-forme d'exécution de services. En effet, contrairement à des solutions de type Pastry[1, 2] nous proposons de séparer la couche de gestion de la grille pair-à-pair, de la couche de localisation et d'exécution de services. Cette spécification tend à être applicable sur toute architecture pair-à-pair.

Comme nous venons de le préciser, nous ne nous focalisons par sur la gestion de l'architecture de la plate-forme pair-à-pair, mais sur l'aspect implantation, déploiement et exécution des requêtes. Nous faisons, dans le cadre de la modélisation, le choix de ne pas nous aligner sur une architecture spécifique d'exécution, mais de spécifier les opérations de manière détachée. Il sera donc tout à fait possible de plonger cette spécification dans les différentes plates-formes d'exécution : .Net, J2EE, Web Services SOAP, Corba, ... Toute combinaison de systèmes pair-à-pair respectant les contraintes de notre modèle pourra ainsi être composée avec toute plate-forme d'exécution de services.

3. Travaux connexe

L'exécution de code distant a été défini de différentes façons. L'appel distant de code à donné lieu au schéma général des RPC(*Remote Procedure Call*)[1]. La spécification *RPC*

1. RFC707, RFC 1057

spécifie les échanges entre un consommateur d'informations (le client) , et la ressource (le serveur) générateur d'informations et de données. On définit ainsi la possibilité du client à effectuer des requêtes auprès d'un service détenu par le serveur, autrement dit d'invoquer des services.
Différentes implémentations qui mettent en œuvre différents langages, protocoles ou systèmes ont été proposées, dont celles : (i) basées sur des bibliothèques systèmes : ONC RPC[2] ; (ii) basées sur des objets à distance : JAVA RMI, Corba[5] ; (iii) basées sur l'utilisation de composants, J2EE, Net - Mono ; (iv) basées sur les services web[12] : XML-RPC, SOAP.

Le concept de RPC a été transféré dans les grilles, soit à partir des bibliothèques permettant la mise en place de solutions techniques, comme Globus[6] (c'est le cas de Ninf-G[13]), soit à partir des solutions directement conçues pour assurer le GRID RPC.
Les solutions proposées sont hiérarchiques, permettent des exécutions distantes mais nécessitent des points de centralisation de la connaissance, comme dans le cas de Ninf-G[13] et DIET[14] dans lesquels les capacités d'exécution s'enregistrent auprès d'un point de centralisation.

Comme dans le cas des exécutions sur le WEB, des solutions d'implantation de services dans les grilles ont été proposés sur de telles architectures, par exemples les auteurs de[9, 10, 11] propose la spécification de la notion de services dans le cadre des grilles. L'ensemble de ces propositions présentent des solutions hiérarchiques dont les architectures ne présentent qu'un degré de dynamicité réduit, et dont les services d'infrastructure, comme la localisation de services, sont des services eux même centralisés.

Dans le cadre d'architecture pair-à-pair qui présente une gestion de la topologie décentralisée, par exemple Pastry[1, 2] propose une organisation en anneau. Mais la spécification de la localisation des ressources et de la gestion de la topologie se trouve intégrée et entrelacée.

4. Modélisation

Afin de définir notre spécification, nous devons définir le modèle d'application. Le modèle que nous définissons est basé sur 3 composants principaux : le réseau, les nœuds qui composent le réseau et la notion de service.

4.1. Le réseau

Le réseau que nous considérons est le *réseau overlay*. Il s'agit en fait, non pas du réseau physique d'interconnexion, mais du réseau logique qui est mis à notre disposition par la plate-forme pair-à-pair. Afin d'être exploitable dans notre spécification, on modélise le réseau sous forme d'un graphe $G(V,E)$ où V est l'ensemble de nœuds et E l'ensemble des liens de communications. Pour chacun des nœuds on définit la notion de voisinage ainsi : deux nœuds i et j sont voisins si

1) i peut communiquer avec j et j peut communiquer avec i

2) i connait j dans sa liste de voisin et j connait i dans sa liste de voisins

Ces définitions permettent la prise en compte des différentes spécificités du réseau : éléments de sécurité comme des *firewall* qui limitent la possibilité de communication ; éléments de configuration et d'implémentation du réseau comme du NAT.

2. RFC 1790

REMARQUE. — On pourra tout de même remarquer que nous nous attachons uniquement à la communication et non à la connexion : deux nœuds qui ne peuvent communiquer de manière directe (tous deux derrière des *firewalls* peuvent être considérés comme voisins s'ils exploitent une file d'échange partagée, un *MOM* (Middleware On Message).

Le réseau offre au minimum les primitives de communication suivantes :

1) émission d'un message

- envoie d'un message à un voisin `send(Neigh, message)`

- envoie d'un message à un nœud de la communauté `route(id,message)`

2) réception d'un message

- réception bloquante d'un message `receive()`

- réception non bloquante d'un message `async_receive(callback)`, ou `callback` est une procédure à exécuter à la réception d'un message

D'autres primitives de communication peuvent être fournies par le réseau, qui peuvent être originales, ou la combinaison des primitives précédentes, par exemple la primitive *diffusion* qui permet de diffuser une information à l'ensemble des nœuds.

4.2. Les nœuds

Chacun des nœuds détient un identifiant unique dans le réseau pair-à-pair. Les nœuds sont en charge de la gestion locale du réseau, ils assurent collectivement les tâches décrites précédemment. Outre ces tâches, ils assurent le réceptacle des services, en fait la plate-forme d'exécution. Les principales charges de cette plate-forme sont : la gestion du déploiement ; le cycle de vie des services ; gestion des requêtes et des exécutions.

Les nœuds détiennent de plus une table des services, qui, comme nous le verrons par la suite, répertorie au minimum l'ensemble des services détenus localement, et au maximum l'ensemble de tous les services présents sur la grille.

4.3. Les services

Les services sont des objets intégrables aux plates-formes d'exécution détenues en chaque nœud. Un service est caractérisé par :

– la plate-forme d'exécution pour laquelle il a été conu ;

– les ressources nécessaires à son exécution (ressources de calcul, ressources de données, ressources de conexion ...) ;

– les données, format et contenu nécessaire lors de l'invocation du service ;

– le format et les contraintes des données résultat.

5. Spécification des services en environnement pair-à-pair

Nous allons définir l'ensemble des primitives et opérations spécifiques à l'exécution de services sur notre grille pair-à-pair.

5.1. Invocation d'un service `invoke`

L'invocation d'un service peut se faire sur n'importe quel nœud de la communauté. Cette invocation peut se faire soit à partir d'un nœud de la grille, soit par un nœud extérieur à la communauté. Dans ce second cas, tout nœud de la communauté peut être le «point

d'invocation», qui sera en charge de faire l'invocation sur la communauté, de récupérer le résultat et de retourner ce dernier, ou une erreur, à la source de la requête.

5.2. Localisation des services `lookup`

La localisation des services est la première étape de la chaîne d'exécution d'un service. Dans le cas de services WEB de type SOAP[4] on peut notamment citer l'utilisation de l'annuaire UDDI[3], dont l'objectif est de repérer le fournisseur du service, et la localisation du service lui même.

Chaque nœud de la grille pair-à-pair détient un tableau `Service` qui stocke, les identificateurs des nœuds de la grille en fonction de leur signature. `lookup` eput être considéré comme un service local, en effet, il est disponible sur chacun des nœuds de la communauté, mais n'a pas vocation a être invoqué depuis une machine autre que la machine locale.

A la réception d'une requête d'exécution de service par un nœud de la plate-forme, ce dernier fait appel à ce service de localisation qui exécute les opérations suivantes :

– si le service est présent sur le nœud, celui-ci est invoqué ;

– si le service n'est pas présent sur le nœud, si le nœud connaît la localisation du service, il route la requête vers le nœud qui l'héberge ;

– si le service n'est pas présent sur le nœud, si le nœud ne connaît pas la localisation du service, il fait suivre la requête vers ses voisins.

Nous verrons par la suite la collecte d'information permettant de remplir le `registre de services`.

REMARQUE. — 1. Le service requis peut être absent de la grille, soit suite à une panne, ou car il n'a pas encore été déployé. Dans ce cas, on pourra utiliser un algorithme de type algorithmique à vague, qui retournera au point d'invocation un code d'erreur qui sera transmis à l'émetteur de la requête

REMARQUE. — 2. Comme nous l'avons précisé précédemment, notre spécification n'est pas liée à une architecture particulière. dans le cas d'une architecture basée sur un anneau orienté ou une chaîne, la procédure `forward` ne permet d'exploiter la communauté que de manière sérielle, c'est à dire un nœud «à la fois». Par contre dans le cadre d'un anneau non orienté, d'un arbre ou d'une structure quelconque la procédure d'exploration initiée par `forward` se fera sous la forme d'une vague, en parallèle.

5.3. Exécution de service et retour des résultats `exec`

Une fois que la requête a atteint le nœud qui détient le service, ce dernier exécute celui-ci. La première étape va consister à identifier les paramètres de l'invocation avec ceux de l'appel. Cette exécution peut provoquer soit la génération d'un résultat, soit la génération d'une erreur. Une fois le résultat produit, ce dernier est routé vers le «point d'invocation».

5.4. Déploiement de service `deploy`

Le déploiement d'un service nécessite une première phase de détection des noeuds de la plate-forme en mesure d'héberger et d'exécuter le service. Un sous ensemble des sites de la plate-forme sera donc candidat à l'hébergement du service. Parmi ces noeuds

il faudra donc, en appliquant une stratégie particulière, désigner l'hébergeur du service. Nous proposons trois stratégies de placement :

1) une stratégie aléatoire qui consistera à tirer au sort le nœud sur lequel sera déployé le service.

2) une stratégie éqilibrée qui consistera à collecter les statistiques des nœuds et à tenter d'équilibrer la charge entre les candidats.

3) la stratégie "premier nœud" qui consistera à déployer le service sur le premier nœud détecté capable de supporter son exécution.

REMARQUE. — La première stratégie implique de connaitre l'ensemble des noeuds candidats au déploiement ; la seconde de connaitre l'ensemble des nœuds candaidats et des informations sur chacun et la troisième limite la connaissance nécessaire, mais peut clairement mener à une surcharge.

5.5. Enregistrement de service

Il est possible d'exploiter la plate-forme par l'intermédiaire de recherche sans mémoire des serevices à chaque invocation. Comme nous l'avons précisé précédemment, nous pouvons augmenter les performances de notre plate-forme en exploitant un registre de services. Nous envisageons la construction de ce registre de services selon deux principales stratégies :

1) la diffusion au déploiement, dans cette première stratégie, la diffusion de l'information sur la localisation du service se fera sous la forme d'une innondation auprès de toutes les neouds de la plate-forme

2) la diffusion à l'appel, dans cette seconde stratégie l'information de localisation sera diffusée lors de la réponse après invocation. Tous les sites relayant la réponse vers le point d'invocation seront alors informé de la localisation du service.

Afin d'éviter la surcharge des mémoires de sites par le registre de service, il est possible de répartir ce registre sur plusieurs sites consécutifs. Ce qui aura l'intérêt de réduire la charge des sites, et de limiter la perte d'information dans le cas de disparition de site.

6. Propositions d'implémentation

Pour implémenter notre solution, nous proposons deux approches : une approche utilisant la topologie P2P en anneau et une approche utilisant la topologie P2P en arbre.

6.1. Topologie P2P en anneau

Cette topologie adopte une architecture P2P distribuée qui essaie de répartir la totalité des fonctions du système entre les nœuds à savoir : la recherche, le routage et la récupération des services dans le réseau logique.

Les protocoles les plus représentatifs utilisant la topologie en anneau sont Chord[15] et Pastry[1], tous deux basés sur les tables de hachage distribuées (DHT).

Avec cette approche, nous utilisons une topologie similaire à celle de Pastry. Ainsi, lorsqu'un message est routé vers un destinataire, le préfixe commun entre les nœuds intermédiaires et la destination augmente à chaque saut. Chaque nœud maintient une table de routage contenant $(2^b - 1) * log_{2^b} N$ voisins où $2^b - 1$ représente le nombre d'entrées de chaque ligne. En plus de la table de routage, ce protocole maintient un ensemble de

voisins virtuels (séquentiels) nommé leaf (feuille), qui contient $L/2$ successeurs et $L/2$ prédécesseurs. Enfin, la dernière table de ce protocole, maintient un ensemble des voisins réels (physique).

Notre approche de recherche/localisation, d'exécution et d'enregistrement de service est décrit comme suit : Lorqu'un nœud cherche un service dans la grille P2P, il regarde d'abord si ce dernier est détenu par un de ses voisins. Si oui, le service est exécuté dans le nœud voisin détenteur. Sinon, il forward la requête de recherche/localisation à son voisin qui a le plus long préfixe dans sa table. Ce nœud voisin exécute le même algorithme, ainsi de suite.
Ainsi, si la requête revient au nœud initiateur, c'est que le service recherché est absent. Par contre, si un nœud localise le service recherché, il l'exécute et renvoie le résultat qui prendra le chemin inverse de celui de localisation. Dans ce chemin de retour, tous les nœuds intermédiaires ainsi que le nœud source mettrons à jour leurs registre de service ($ID_Service$, ID_Node).

6.2. Topologies P2P en arbre

Cette topologie adopte une architecture P2P décentralisée qui repose sur des interconnexions de super nœuds sur le niveau haut de la hiérarchie.

Pour implémenter cette approche de topologie P2P en arbre de notre spécification, nous nous basons principalement sur le protocole Gnutella2[16] qui propose d'organiser le réseau logique selon une architecture décentralisée sur deux niveaux. Les nœuds simples ou feuilles se rattachent à un super nœud et les super nœuds entre eux sont reliés dans un réseau Gnutella1[17]. Avec cette approche, les services partagés par une feuille sont enregistrés sur le super nœud responsable de cette feuille.
Un nœud peut être nommé super nœud suivant plusieurs critères : cela peut dépendre de son adresse (publique ou privée), de son système d'exploitation, de sa bande passante, de l'instant depuis lequel il est connecté au réseau, ou de ses ressources matérielles (puissance de calcul, capacité mémoire).

Nous proposons l'approche de recherche/localisation, d'exécution et d'enregistrement de service suivante : Lorsqu'un nœud recherche un service, il envoie sa requête à son super nœud. Celui-ci effectue alors la recherche parmi les services des nœuds qui lui sont rattachés (recherche locale). Si le service est localisé, alors il l'exécute et renvoie le résultat. Sinon, la requête de localisation est diffusée aux autres supers nœuds. Ainsi, si aucune reponse n'est retournée après un temps T_wait, c'est que le service recherché est absent. Sinon, le super nœud détenant le service recherché, l'exécute et renvoie le résultat au super nœud source. Ce dernier renvoie le résultat à son nœud source et à jour son registre de service ($ID_Service$, ID_Node).

7. Conclusions et perspectives

Au constat que les grilles exploitant la notion de services sont basées sur des architectures hiérarchiques fortement centralisées, nous proposons une solution originale de spécification en couche qui tente de séparer la gestion de l'architecture et du réseau p2p de la couche d'exécution. La spécification est non liée à un réseau pair-à-pair particulier ou à un protocole de gestion de services défini à l'avance.

Comme perspective immédiate, nous mettrons en oeuvre nos deux solutions d'implémentation. Ce travail sera suiva par une étude de solution de tolérance aux fautes dont le redéploiement dynamique de services en cas de détection de requêtes non résolues.

Parallélement, le développement des démonstrateurs et une comparaison des stratégies de localisétion des services et des informations sur les services sont envisagés.

8. Bibliographie

[1] A. ROWSTRON, P. DRUSCHEL, « Pastry : Scalable, distributed object location and routing for large-scale peer-to-peer systems », *In Proc. IFIP/ACM Middleware 2001, Heidelberg, Germany*, Nov. 2001.

[2] M. CASTRO, P. DRUSCHEL, A-M. KERMARREC, A. ROWSTRON « One ring to rule them all : Service discovery and binding in structured peer-to-peer overlay networks », *SIGOPS European Workshop, France*, Septembre, 2002.

[3] LUC CLEMENT, SYSTINET, ANDREW HATELY, IBM, CLAUS VON RIEGEN, SAP AG, TONY ROGERS, COMPUTER ASSOCIATES « UDDI v.3.0.2. OASIS Specification », October 2004.

[4] MARTIN GUDGIN, MARC HADLEY, JEAN - JACQUES MOREAU, HENRIK FRYSTYK NIELSEN « Simple Object Access Protocol (SOAP) Version 1.2. W3C », Juillet 2001.

[5] JEAN-MARC GEIB, CHRISTOPHE GRANSART, PHILIPPE MERLE « Corba. Des concepts à la pratique », 1999.

[6] IAN FOSTER « Globus Toolkit version 4 : software for service oriented stsems », *JSCT*, vol. 21 n° 4 Juillet 2006.

[7] I. FOSTER, C. KESSELMAN, S. TUECKE « The anatomy of the grid : enabling scalable virtual organizations », *IJSA'01*, vol. 3 2001.

[8] O. FLAUZAC, M. KRAJECKI, L.A. STEFFENEL « CONFIIT a middleware for peer-to-peer computing », *Journal of supercomputing*, vol. 53 n° 1 Juillet 2010.

[9] S TUECKE, K. CZAIKOWSKI, I. FOSTER, J. FREY, S. GRAHAM, C. KESSELMAN, D. SNELLING, P. VANDERBILT « Open grid services infrastructure »,

[10] S TUECKE, K. CZAIKOWSKI, I. FOSTER, J. FREY, G. CARL « Grid service specification », 2002.

[11] D. TALIA « The open grid services architecture : Where the grid meets the web », *IEEE Internet Computing*, vol. 6 n° 6 2002.

[12] W.A. NAGUY, F. CURBERA, S. WEERAWARANA « Web services : Why and how ? », *ACM OOPSLA, Workshop on Object-Oriented Web Services*, 2001.

[13] YOSHIO TANAKA, HIDEMOTO NAKADA, SATOSHI SEKIGUCHI, TOYOTARO SUZUMURA, SATOSHI MATSUOKA « Ninf-G : A Reference Implementation of RPC-based Programming Middleware for Grid Computing », *Journal of Grid Computing*, 2003.

[14] EDDY CARON, FRÉRÉRIC DESPREZ « DIET : A Scalable Toolbox to Build Network Enabled Servers on the Grid », *International Journal of High Performance Computing Applications*, vol. 20 n° 3 2006.

[15] I. STOICA, R. MORRIS, D. KARGER, M.F. KAASHOEK , H. BALAKRISHNAN « Chord : A Scalable Peer-to-Peer Lookup Service for Internet Applications. », *In Proceedings of SIGCOMM. San Deigo, CA*, 2001.

[16] GNUTELLA 0.6 « The Gnutella Protocol Development. [En ligne] http ://rfc-gnutella.sourceforge.net/developer/index.html », July 2003.

[17] GNUTELLA 0.4 « The Gnutella Protocol Specification. [En ligne] http ://rfc-gnutella.sourceforge.net/developer/stable/index.html », 2001.