Université de Nice — Sophia Antipolis
Master Informatique 1re année

Travail d'Étude et de Recherche
Mars–Juin 2009

DiscoGRID

Expérimentations d'applications sur grilles hétérogènes

Étudiants
Arnaud Gastinel
Cyprien Nicolas

Encadrants
Françoise Baude
Elton Mathias

Table des matières

Chapitre 1

Présentation du sujet

1.1 Cadre de travail

Le TER s'est déroulé au sein de l'équipe de recherche OASIS [1] [CAR 99], projet commun entre le laboratoire I3S [2] (UMR6070), commun entre le CNRS et l'UNSA, et l'INRIA Sophia Antipolis.

Nous travaillions sur le site de l'INRIA, dans le bâtiment Lagrange.

Nous étions encadrés par M^me^ Françoise Baude, Maître de Conférence à l'UNSA, et M. Elton NICOLETTI MATHIAS, doctorant.

Introduction

Le sujet a été originalement donné en anglais, les sections suivantes sont donc des traductions des informations données accessible à l'adresse suivante, en anglais :

`http://deptinfo.unice.fr/~huet/M1INFO/08-09/TER/z_z_Discogrid.html`.

1.2 Contexte Scientifique

Le travail demandé est assez expérimental. Le premier objectif est d'être capable de mener des tests expérimentaux intensifs d'une application parallèle et distribuée déjà existante, sur une large variété d'infrastructures de calculs. Cette application scientifique et parallèle simule numériquement la propagation des ondes électromagnétiques.

L'exécutable a été développée dans le cadre du projet DISCOGRID [DIS 05], prenant fin mi-2009. Il a été financé par l'ANR [3].

Il est basé sur les composants GCM [BAU 08] offerts par PROACTIVE [PRO 99], ainsi qu'une bibliothèque C++ pour envelopper les processus natifs écrits en FORTRAN.

La dernière étape du projet consiste à obtenir des résultats montrant la réelle efficacité de l'approche SPMD [4] hiérarchique originelle de la programmation distribuée. Elle est plus facile à utiliser qu'une application écrite purement en FORTRAN et MPI [5], montrant des

[1] Objets Actifs, Sémantique, Internet et Sécurité

[2] Laboratoire d'Informatique, Signaux et Systèmes de Sophia-Antipolis, `http://www.i3s.unice.fr/I3S/`

[3] Agence Nationale de la Recherche, `http://www.agence-nationale-recherche.fr/` (fr)

[4] Single Process, Multiple Data, `http://en.wikipedia.org/wiki/SPMD` (en)

[5] Message Passing Interface, `http://en.wikipedia.org/wiki/Message_Passing_Interface` (en)

performances équivalentes par rapport à une implémentation de MPI spécifique aux grilles de calcul, et offrant plus de flexibilité.

1.3 Description du travail

1. Étudier le framework ProActive et comprendre ces concepts fondamentaux,
2. Étudier le framework DiscoGRID existant, et l'application SPMD de simulation parallèle,
3. Développer une nouvelle version, en utilisant Java nio [6], afin d'optimiser les communications entre Java et le code natif,
4. Exécuter les anciennes et nouvelles applications, en priorité sur Grid'5000 [GRI 05],
5. Comparer les performances lors de l'exécution de l'implémentation écrite purement en Fortran et MPI, avec les bibliothèques MPICH [7] et GridMPI [8],
6. Étudier comment soumettre des tâches sur egee [9] en utilisant ProActive,
7. Étudier comment soumettre des tâches sur Amazon ec2 [10] en utilisant ProActive,
8. Étudier comment soumettre des tâches sur ProActive « p2p desktop grid »,
9. Identifier et résoudre les problèmes qui pourraient survenir lors de l'utilisation combinée des machines venant de ces infrastructures.

1.4 Restriction du sujet et répartition des tâches

Le sujet étant trop ambitieux pour la durée de notre TER, les points 6 à 8 n'ont donc pas été effectués. Un autre TER s'est occuper de consolider les points 6,7(surtout) et 8, et Arnaud continuera sur ces points là avec DiscoGRID lors de la poursuite en stage à l'issue du TER.

Nos encadrants nous ont demandé, pour Arnaud par se familiariser avec Grid'5000, avant d'essayer de déployer une application DiscoGRID sur plusieurs ressources distribuées (Grid'5000 et Amazon ec2), puis de résoudre le problèmes des communications ProActive entre différents réseaux. Cyprien doit étudier l'implémentation des communications internes avec IPC, et les optimiser, notamment en utilisant Java nio. Lors de l'expérimentation, nous avons simulé une grille, en utilisant les Eons [11].

Un autre sujet de TER était commun au nôtre concernant la soumissions de tâches depuis ProActive, ces étudiants ce sont notamment intéressés à Amazon ec2, tandis que nous nous sommes concentrés sur le projet DiscoGRID.

[6] New I/O, introduite avec Java 1.4
[7] `http://www.mcs.anl.gov/research/projects/mpi/` (en)
[8] `http://www.gridmpi.org/index.jsp` (en)
[9] Enabling Grids for E-sciencE, `http://www.eu-egee.org` (en)
[10] Amazon Elastic Compute Could, `http://aws.amazon.com/ec2/` (en)
[11] grappe accessible au réseau de production de l'INRIA Sophia

Chapitre 2

État de l'art

2.1 DiscoGRID

2.1.1 Présentation

La description suivante est la traduction du texte de présentation disponible sur la page personnelle de Stéphane Lanteri*, dont l'adresse est donnée en note dans la section 1.2 page 4.*

Le projet DiscoGRID de l'ANR [1] a pour but de proposer un nouveau paradigme pour programmer des applications scientifiques aisément parallélisables sur des environnements de calcul distribués et hétérogènes.

Ces applications modélisent la propagation des ondes électromagnétiques ou les écoulements des fluides, et nécessitent de résoudre des systèmes d'équations aux dérivées partielles [2]. Les solutions s'obtiennent par les méthodes dites des « éléments finis [3] » ou des « volumes finis [4] », car elles sont facilement utilisables avec des maillages non structurés.

Le projet DiscoGRID se concentre donc sur les noyaux numériques de calcul et les difficultés qui en découlent, dans la mesure où elles ont un intérêt pour une large variété d'autres contextes d'exécution. L'accent est mis sur la conception d'algorithmes numériques distribués et la programmation d'un logiciel de simulation qui exploite efficacement les grilles de calculs, et en particulier la plate-forme d'essai Grid'5000.

Enfin, le projet DiscoGRID étudie le caractère applicable des principes modernes du calcul distribué, ainsi que les méthodologies pour le développement de logiciels de simulations de haute performance.

2.1.2 ProActive

DiscoGRID se base sur ProActive, un « middleware » dédié à la programmation parallèle, distribuée et multi-cœur ; distribuée sous licence libre [BAD 06]. Il est implémenté en Java afin d'être facilement utilisable sur les différentes architectures que l'on peut trouver sur des grilles. Il permet de déployer facilement une application sur plusieurs machines, à l'aide de fichiers XML assez simples, que ce soit localement, où bien sur des grilles à travers SSH par exemple.

[1] Référence ANR-05-CIGC-005 ; démarré en janvier 2006

[2] `http://fr.wikipedia.org/wiki/%C3%89quation_diff%C3%A9rentielle_partielle` (fr)

[3] `http://fr.wikipedia.org/wiki/M%C3%A9thode_des_%C3%A9l%C3%A9ments_finis` (fr)

[4] `http://fr.wikipedia.org/wiki/M%C3%A9thode_des_volumes_finis` (fr)

ProActive est centré sur la notion d'activité, réalisée par des objets actifs, qui consitue un couple entre un objet Java et une seule thread. Ils sont capables de s'échanger des messages et de migrer d'un nœud à un autre. [CAR 05] reprend les concepts des Objets Actifs et montre certains propriétés sur eux.

ProActive permet également d'utiliser la programmation orientée composants en proposant les composants GCM construits au-dessus de ceux proposés par Fractal [5]. Fractal et ProActive sont tous deux projets du consortium Object Web [6], qui vise à rassembler une commnauté autour de solutions « Open Source » pour la programmation distribuée.

2.1.3 Partitionneur hiérarchique

DiscoGRID contient un partitionneur qui tente de minimiser les dépendances entre les blocs de données ainsi découpés pour limiter ensuite les communications entre les processus locaclisés dans des grappes différentes.

L'aspect hiérarchique concerne la topologie réseau de la grille, qui détermine les étapes de découpage : premièrement un découpage est fait en autant de parties qu'il y a de grappes dans la grille de déploiement, puis chaque partie est à nouveau découpée en fonction du nombre de nœuds dans chaque grappe.

2.1.4 Exécution et déploiement

Les scripts d'exécution nécessitent plusieurs arguments, afin de spécifier où et comment l'application sera déployée, nombre de grappes (clusters), nombre de nœuds par grappe, les noms des sites et des grappes, et enfin le nom de la grille qui contient ces grappes, tous ces termes et les notions associées sont explicités dans la section 2.2 page 9.

Lors de l'exécution d'une application utilisant DiscoGRID, plusieurs étapes distinctes sont observables :

1. Génération des descripteurs de topologie (éventuellement) Les descripteurs ne sont générés que dans le cas d'un déploiement sur des grilles avec réservation, car le nom des nœuds ne sont pas connus à l'avance ; au contraire des grappes locales, comme les éons, où du fait que les nœuds sont accessibles directement et sans réservation, les fichiers sont déjà écrits.

2. Partitionnement des données Les données sont partitionnées hiérarchiquement, d'abord divisées en part égales pour chaque grappe, puis à nouveaux en autant de parties que de nœud par grappe.

3. Démarrage de ProActive avec les descripteurs Exécution de ProActive, le « Native Starter », et chargement du descripteur de déploiement éventuellement générés. C'est à cette étapes que les fichiers nécessaires sont transférés vers les différents nœuds de déploiement.

4. Déploiement des processus ProActive sur chacun des nœuds, et des services associés Déploiement des processus ProActive sur chacune des machines du descripteur, ainsi que des processus maîtres par grappe.

[5] `http://fractal.ow2.org/` (en)
[6] `http://www.ow2.org` (en)

5. Chargement des bibliothèques de communication de ProActive Le code natif, emballé dans un objet ProActive via JNI, est chargé par les processus ProActive. C'est notamment ce code qui est en charge des communications internes.

6. Inscription auprès du processus ProActive maître de l'état prêt de chacun des processus déployés Une fois que chaque processus ProActive est prêt, il notifie son processus maître local, qui notifiera le « Native Starter » une fois que tous ces subordonnées seront prêts.

7. Exécution des processus Fortran/MPI Le « Native Starter » lance alors l'exécution localement de `mpirun` avec la liste des nœuds générée lors de l'étape 1, afin de démarrer tous les processus Fortran/MPI.

8. Attente du processus ProActive maître de l'initialisation des processus Une fois que chaque processus natif a fait l'appel à `MPI_Init`, les processus maîtres vont construire les structures de communication locales et externes, afin de savoir router les messages correctement par la suite.

9. Lecture des données, et démarrage de l'exécution Les processus Fortran/MPI procèdent à la lecture des maillages précédemment partitionnés, et démarrent le calcul.

10. Attente du processus ProActive maître de la terminaison des processus déployés. L'exécution est terminée, les processus Fortran/MPI, à travers la bibliothèque MPI (`MPI_Finalize`), notifient les processus maîtres, chacun terminant les processus ProActive associés, et ainsi de suite en cascade.

2.1.5 Gestion des communications

Les processus Fortran/MPI sont donc situés sur des nœuds différentes, et ils communiquent à l'aide d'appels à la bibliothèque MPI du système. De façon à être le moins intrusif possible, l'environnement DiscoGRID substitue sa propre bibliothèque MPI à celle du système au moment de la compilation de l'application native.

Ainsi, lors d'un appel à une des procédures MPI (`MPI_Send` par exemple), l'appel passera par la bibliothèque DiscoGRID, qui, suivant la destination (en fonction du numéro de grappe), enverra soit directement le message à travers la bibliothèque standard MPI (communications locales). Sinon il sera transmis à travers la bibliothèque de communication présente dans ProActive, qui s'occupera de transmettre le message au processus ProActive déployé sur le même nœud que le processus natif ; ce sont les communications internes.

Enfin, si le destinataire se trouve sur une autre grappe, ou sur une autre grille, le message sera envoyé à travers le système ProActive au processus maître de la grappe ou de la grille qui supervise le processus distant ; ce sont les communications externes. Une fois le message arrivé à destination, si possible, il est transmis directement au destinataire, sinon il descend dans la hiérarchie des processus ProActive, jusqu'à atteindre celui qui est couplé au processus natif, qui transmettra le message par une communication interne.

2.2 Grilles de calcul

Le terme grappe (cluster) désigne une architecture constitué d'un ensemble d'ordinateur homogènes relié entre eux, généralement par une connexion réseau offrant un très haut débit de communication (Myrinet [7], InfiniBand [8], ...) pouvant atteindre 40 Gigabit/s. Les grappes sont une alternative aux super-ordinateurs, en fournissant une grande puissance de calcul à un prix relativement modeste. Chaque machine de la grappe est appelé un noeud, et sont généralement rangé dans des racks. L'ensemble doit se rapprocher d'une entité de calcul unique : possibilité d'accès aux fichiers sur chaque noeud (par exemple en utilisant une solution NFS).

Lorsque plusieurs grappes sont reliés entre elles, on parle alors de Grille de calcul. Au sein d'une même grappe, les ressources sont fréquemment homogènes contrairement à la grille.

2.2.1 Eons

La grappe de calcul nommé Eon est situé dans les locaux de l'INRIA à Sophia, et elle est dédiée à l'équipe Oasis. Chaque noeud dispose de deux processeurs contenant chacun quatre coeurs chacun cadencé à 2 Ghz. Coté mémoire vive, chaque machine dispose de 16 Go de RAM. Les noeuds sont nommés en suivant le schéma suivant : eon[1-20]. Les eons sont accessibles depuis l'extérieur en passant par la passerelle ssh-sop.inria.fr en utilisant le protocole SSH sur le port 22.

2.2.2 Grid'5000

La grille de calcul Grid'5000 est un projet de recherche scientifique initié par l'INRIA et le CNRS, en 2005, mettant à disposition [9] 1597 noeuds ou 3202 processeurs ou encore 5714 coeurs. Les noeuds sont réparties en neuf sites en France, :

Bordeaux	Grenoble	Lille
Lyon	Nancy	Orsay
Rennes	Sophia-Antipolis	Toulouse

Cette grille de calcul est la première du genre en Europe et fourni aux chercheurs Français une excellent support pour le calcul distribué.

L'accès à Grid'5000 depuis l'extérieur se fait en utilisant la passerelle [10] de chaque site (exemple : acces.sophia.grid5000.fr) accessible par le protocole SSH sur le port 22.

2.3 Communication multi-grappes / multi-grilles

Le projet DiscoGRID ciblait initialement des applications sur la grille Grid'5000, les communications dites externes étaient réduites à des communications inter-grappes, chaque grappe (ainsi que chaque site) étant relié par le réseau national RENATER [11], le framework

[7] `http://www.myricom.com/myrinet/overview/` (en)

[8] `http://www.infinibandta.org/` (en)

[9] à ce jour (mai 2009)

[10] Souvent appelée « machine frontale », ou « frontend »

[11] Le REseau NAtional de télécommunications pour la Technologie l'Enseignement et la Recherche `http://www.renater.fr/`

ProActive ne souffrait d'aucune restriction réseau. Cette partie du travail porte ainsi sur une extension de DiscoGRID permettant la communication entre plusieurs grilles.

Comme expliqué dans la section 2.1.5 page 8, lorsque le destinataire d'un message se trouve sur une autre grappe (ou sur une autre grille), le framework DiscoGRID s'appuie sur ProActive.

Nous allons détailler les avantages ainsi que les inconvénients des protocoles de communication utilisés dans ProActive.

2.3.1 Le protocole RMI

« Remote Method Invocation » (RMI) est un méchanisme Java (API + Environnement), développé par Sun, permettant les appels de méthodes sur des machines distantes. Elle s'appuie sur le protocole « Java Remote Method Protocol » (JRMP) pour communiquer via une connexion TCP/IP. On dénommera par la suite, par abus de langage, RMI comme étant le protocole de communication complet utilisé par défaut dans ProActive.

Fonctionnement Le fonctionnement global de RMI peut être succinctement décrit de la façon suivante :

Étape 1 Un `rmiregistry` est lancé sur la machine hébergeant l'objet sur lequel on souhaite appeler une/des méthode(s) depuis une JVM distante. Le registre RMI garde une référence de tous les objets qui s'y sont précédemment enregistrés.

Étape 2 Pour pouvoir être accessible de l'extérieur, un objet devra étendre la classe `UnicastRemoteObject` (paquetage `java.rmi.server`), et implémenter une interface qui étend elle-même l'interface `java.rmi.Remote`. Ensuite cet objet devra s'enregistrer auprès du registre RMI. Une référence distante est alors créée permettant de joindre directement l'objet au moyen d'une connexion TCP/IP classique (toujours en utilisant le protocole JRMP).

Étape 3 Lorsque l'on souhaitera appeler une méthode distante sur un objet distant, il devra disposer d'une référence distante pour le contacter. Cette référence peut être récupérée en envoyant une requête au registre RMI. Cette référence permettra donc de contacter directement l'objet à l'avenir, mais nécessite un accès réseau (port ouvert, adresse publique, ...).

C'est donc pour cela que le protocole RMI pose des problèmes dans les communications inter grilles, car celles-ci, pour des raisons évidentes de sécurité, sont fréquemment coupées du reste du réseau (typiquement, le réseau Internet, ou bien le réseau RENATER) par un pare-feu. Le seul moyen d'accéder à ces grilles est de se connecter d'abord à une passerelle via le protocole SSH, cette passerelle faisant partie de la grille et ayant alors accès à tous ses nœuds.

2.3.2 Le protocole PAMR

Notre première piste a été d'étudier le protocole « ProActive Message Routing » PAMR [12] maintenu par ActiveEon [CAR 07], la start-up fondée par Denis Caromel visant à développer intensivement ProActive. Le but du protocole PAMR est de permettre au framework

[12] `http://proactive.inria.fr/release-doc/pa/multiple_html/pamr.html`

ProActive de communiquer au sein d'une architecture utilisant un pare-feu ou un routeur NAT[13].

Fonctionnement Le fonctionnement de pamr se base sur l'utilisation d'un routeur applicatif, exécuté au moyen du script bin/startRouter(.sh/.bat). L'utilisation du protocole pamr au sein du framework ProActive se fait à l'aide des options de configuration[14] :

- `proactive.communication.protocol`
- `proactive.net.router.address`
- `proactive.net.router.port`

Le premier devant avoir la valeur « pamr », les deux autres servant à préciser, respectivement, l'adresse ip et le port à utiliser pour contacter le routeur pamr. La seule restriction réseau pour pouvoir utiliser le protocole pamr, est que toutes les machines doivent pouvoir accéder au routeur au moyen d'une connexion tcp sortante.

Pour s'appliquer à notre cas d'utilisation, il suffirait d'utiliser une redirection de port par le protocole ssh (appelé à tort tunnel ssh) à partir d'une machine (que nous appellerons M1) de la grille (A) vers la machine (M2) de l'autre grille (B) sur laquelle est exécuté le routeur pamr. Il faudra alors spécifier pour la grille *A* que le routeur pamr est situé sur *M1*.

Après avoir approfondi l'étude du fonctionnement de pamr, il a semblé évident, qu'il était inapplicable dans notre cas d'utilisation pour les raisons suivantes :

- Les connexions ne sont plus point à point mais passent toutes par le routeur pamr, même au sein d'une même grappe, ce qui nuit gravement aux performances du framework DiscoGRID qui s'efforce de minimiser les communications externes en maximisant les communications internes au moment du partitionnement.
- Le routeur ne fonctionnant pas en flux de données, une requête doit être transmise entièrement avant d'être « routée » vers sa destination.

2.3.3 Le protocol rmi+ssh

La dernière piste que nous avons explorée fut celle du protocole rmi+ssh. Partant du fait que les passerelles des différentes grilles ne sont accessibles que par le protocole ssh, nous avons commencé à étudier le protocole précédemment nommé, pour pouvoir l'adapter à notre cas d'utilisation.

Fonctionnement L'implémentation de ssh utilisée est celle fournie par la société Trilead [15] sous le nom ganymed. Elle est écrite entièrement en Java et délivrée sous licence libre compatible avec la Licence Publique Générale gnu gpl [16].

L'utilisation du protocole rmi+ssh au sein du framework ProActive se fait en spécifiant comme précédemment des options de configuration de ProActive :
`proactive.communication.protocol` avec pour valeur `rmissh`.

[13] Network Address Translation `http://fr.wikipedia.org/wiki/Network_address_translation`

[14]. Deux méthodes possibles, soit en passant par un fichier de configuration en XML, soit en spécifiant des arguments (-Dargument) à la machine virtuelle au moment de l'exécution

[15] `http://www.trilead.com/SSH_Library/`

[16] `http://www.gnu.org/licenses/gpl.html`

Le fonctionnement global est d'établir une redirection de port en utilisant SSH depuis un port local quelconque vers le port désiré (port d'écoute du registre RMI ou bien port d'écoute de l'objet distant au sens de RMI) sur la machine distante hébergeant l'objet (ou le registre RMI contenant une référence vers l'objet distant). Ensuite le protocole de communication RMI sera utilisé en se connectant sur l'hôte local et sur le port choisi.

De nombreuses options de configuration peuvent être alors spécifiées de la même façon, pour plus de précision, se référer à la documentation officielle ProActive.[17]

Une option intéressante, qui mérite d'être citée dans le présent document est : `proactive.tunneling.try_normal_first`, qui lorsqu'elle a la valeur `true`, permet au framework ProActive de d'abord essayer d'utiliser le protocole de communication RMI, puis, si la connexion n'est pas possible, alors le protocole RMI+SSH sera utilisé.

2.4 Communications internes

Cette tâche consitait plus particulièrement à considérer les communications internes, et à les optimiser en utilisant l'API offerte par le paquetage `java.nio`.

Nous avons du pour cela étudier l'implémentation actuelle, basée sur la Java Native Interface (JNI) et les Inter Process Communication (IPC) d'Unix, qui est reprise dans le noyau Linux.

2.4.1 Java Native Interface

JNI est une possibilité offerte par Java pour exécuter du code natif, et elle a été introduite avec le Java Developpement Kit (JDK) 1.2. Au niveau du code Java, deux nouvelles possibilités apparaissent : d'une part l'ajout du mot clef « native » lors de la déclaration du nom de la méthode, et la classe `java.lang.System` possède une méthode statique `loadLibrary(String)` qui permet de charger dynamiquement une bibliothèque système.

L'outil `javah`, fourni par le JDK, prend en paramètre les noms des classes qui contiennent des déclarations de méthodes natives et génère les fichiers d'en-tête C/C++ qui contiennent les déclarations de ces méthodes pour le code natif. Il ne reste plus au programmeur qu'à écrire les définitions de ces fonctions.

Il suffit ensuite de compiler les fichiers contenant les déclarations de ces fonctions en tant que bibliothèque dynamique, afin qu'elle puissent être chargées pendant l'exécution à l'aide de la méthode `System.loadLibrary()`.

2.4.2 InterProcess Communication

Les IPC représentent un ensemble de fonctions, introduit à l'origine par les environnements Unix System V, qui permettent à des processus différents de communiquer à travers le noyau du système ; les principales méthodes sont listées ci-dessous :

- Tubes (Pipes, ou FIFOs s'ils sont nommés)
- Queues de messages
- Sémaphores
- Segments de mémoire partagée

[17] `http://proactive.inria.fr/release-doc/pa/multiple_html/SSHTunneling.html`

Dans DiscoGRID, l'implémentation utilisait des queues pour pouvoir échanger des messages entre les processus natifs et ProActive. Il faut donc maintenir une queue pour les messages envoyés par ProActive à destination du processus natif, et une autre pour les messages retour.

Il était connu que l'implémentation actuelle était peu performante, notamment au niveau de la taille maximale de la queue de messages, qui ne pouvait être modifiée qu'en procédant à une nouvelle compilation du noyau du système, cette opération n'étant pas envisageable sur les machines d'une grille de calcul comme Grid'5000.

2.4.3 Problèmes liées à l'implémentation actuelle

Ces queues de messages ont plusieurs défauts, principalement dus au fait que ce concept introduit dans les noyaux Unix est très ancien, et l'implémentation n'a pas évolué.

Limites Les queues sont limitées sur plusieurs points : premièrement, leur nombre. En effet, sous les systèmes Fedora Linux, le nombre maximal de queues existantes sur le système est limité à seize. Ensuite, une queue ne peut contenir au maximum que seize kiloctets (16 ko) de données, et un message ne peut excéder huit kiloctets (8 ko), ces deux dernières limites sont généralement communes à tous les systèmes.

Pour essayer de contourner ces très faibles limites, l'implémentation de Vincent Cavé présente dans DiscoGRID utilise deux queues de messages pour l'envoi de message de ProActive vers le processus natif, et inversement. Il a également implémenté un algorithme de découpage des messages dans le cas où leur taille dépasse les huit kiloctets.

Accès concurrents La queue de message étant entièrement partagée entre les deux processus, il est nécessaire d'utiliser des sémaphores pour garantir une exclusion mutuelle lors des accès en écriture et en lecture à la queue de message, notamment dans le cas où les messages ont été découpés.

Lisibilité Ce dernier point n'est pas lié aux IPC, mais plutôt à la manière dont le code a été écrit. En effet, il n'y a qu'un seul fichier source, qui est compilé et embarqué dans une librairie qui est chargée par les deux processus communicants. Le code étant le même pour les procédures d'envoi et de réception des messages, les variables contenant les descripteurs des queues n'ont pas les mêmes valeurs selon que le code est exécuté par le processus ProActive ou par le processus natif.

Chapitre 3

Travail effectué

3.1 Communications internes

La présente implémentation n'étant pas optimale, nous avons travaillé à améliorer les communications internes par une nouvelle approche.

3.1.1 Nouvelles idées d'implémentation

Notre encadrant nous a suggéré plusieurs points pour réfléchir à une nouvelle implémentation : Sockets, l'API Java NIO, et éventuellement XML/RPC.

Nous avons d'abord essayé avec les Sockets Unix, qui ont l'avantage d'être plus facile à utiliser que les Sockets Inet, car elles sont uniquement locales, et peuvent être éventuellement accédées à travers un fichier sur le disque.

Cette possibilité à vite été abandonnée, car il faut offrir une solution pérenne et portable, afin que DiscoGRID puisse être utilisé sur les plate-formes Win32 par exemple. Nous nous me sommes donc intéressé aux Sockets TCP, en suivant la structure du code de l'implémentation IPC.

3.1.2 Implémentation TCP

Remarques sur l'implémentation

L'implémentation a été plutôt rapide, et à entièrement été faite en `C`. Il a ensuite fallu adapter les scripts et fichiers de compilation de DiscoGRID et de ProActive afin de permettre à l'utilisateur, au moment de la compilation de l'application native avec DiscoGRID, de choisir si les communications passeraient par les IPC ou par TCP.

TCP pose le problème du choix d'un numéro de port commun pour les deux processus, à cette fin, une directive de compilation pour le script Ant a été ajoutée afin de pouvoir définir un autre port au moment de la compilation.

Enfin, il est nécessaire, pour établir la connexion, qu'un processus joue le rôle de serveur, et l'autre de client. L'ordre des étapes dans le processus de déploiement ne m'ont pas fait hésiter sur ce point : le processus ProActive s'occupe de créer la socket, d'appeler les procédures de liaison à la carte réseau (`bind`), et d'écoute (`listen`) de connexions entrantes. Le processus natif, n'a plus qu'à faire une demande de connexion (`connect`) sur la socket.

L'établissement de la connexion étant une étape bloquante avec des sockets, il était nécessaire de retarder le plus possible l'acceptation de la demande de connexion (`accept`) afin que le processus ProActive ne soit pas bloqué.

En effet, s'il l'était, il ne pourrait notifier son processus superviseur qu'il est prêt, et donc ce dernier ne pourrait pas démarrer l'exécution des processus natifs. C'est donc lors du premier appel du processus ProActive à l'une des méthodes d'envoi ou de réception de message que l'acceptation sera faite. Ce choix est justifié car les méthodes d'envoi et de réception sont intrinsèquement bloquantes.

Avantages

Il existe des avantages significatifs à utiliser TCP au lieu d'IPC pour les communications internes. Les sockets permettent des échanges bidirectionnels, c'est-à-dire qu'il n'est pas nécessaire d'ouvrir deux sockets pour permettre l'échange de message dans les deux directions.

Les limites systèmes sur le nombre de sockets ouvertes sont également plus larges, Linux en autorise 1024 simultanément, donc en utiliser deux [1] ne pose pas de problème de disponibilité de ressources, au contraire de l'utilisation de quatre queues de message sur seize.

Les tampons d'envoi et de réception des sockets ont une capacité respective de cent kiloctets (100 ko), ce qui laisse bien plus de place pour stocker les messages.

De plus, il n'y a plus besoin de garantir une exclusion mutuelle, du fait que lors de l'envoi d'un message, le message sera transféré dans le tampon d'envoi, puis traité par le noyau pour être transmis dans le tampon de réception du processus destinataire.

Enfin, l'ordre premier arrivé, premier servi (FIFO) est préservé par construction, car contrairement à l'implémentation IPC, on utilise qu'une seule socket au lieu de plusieurs queues ; ainsi que par l'utilisation du protocole TCP.

Inconvénients

Outre le problème du numéro de port, qui doit pouvoir être libre afin que les deux processus puissent communiquer, il y a le problème des copies multiples de messages.

Avec l'implémentation IPC, le message est copié dans la queue par le processus émetteur, puis lu depuis cette même queue par le destinataire. Pour TCP, il y a une copie supplémentaire du message, depuis le tampon d'envoi vers le tampon de réception de l'autre processus, cette copie est effectuée par le noyau et ne peut donc être évitée.

Il existe cependant un autre endroit où le message est copié, et qui peut être contourné. Les appels effectués par Java au code de communication passent par JNI, or, l'argument de l'appel de la méthode est un objet Java, en l'occurrence, un tableau d'octets (`byte[]`), il faut donc dans le code JNI accéder aux éléments de ce tableau et le copier dans un type primitif de C, à savoir un tableau de caractères (`char[]`). Bien qu'il soit théoriquement possible de demander à JNI de ne pas effectuer de copie, l'API ignore[2] cette demande et effectue quand bien même une copie des données.

La seconde option d'implémentation suggérait l'utilisation de Java nio, nous avons ajouté une nouvelle classe pour utiliser cette API.

[1] il faut en compter une par extrémité (end-point), soit une pour chaque processus

[2] Ceci à été vérifié dans l'implémentation de OpenJDK

3.1.3 Modification d'implémentation : Java nio

La nouvelle API pour gérer les entrées / sorties définit les classes `java.nio.*Buffer` qui peuvent être passées directement en tant qu'argument à des opérations de lecture ou d'écriture. Le sous paquetage `channels` définit des nouvelles structures pour effectuer des opérations sur les fichiers et sockets. Cette API est plus performante que l'API traditionnelle, `java.io` à en croire [PUG 03].

La fonction d'envoi de message prend un tableau d'octets en argument, mais il peut être enveloppé dans un `ByteBuffer` à l'aide de la méthode `wrap()`. Cette méthode garantit que le tableau d'octet sert directement de support au `Buffer` (*backing array*).

La socket peut être ouverte à l'aide de la méthode `ServerSocketChannel.open()`, et accepte de recevoir des `ByteBuffer` en argument pour les méthodes `read()` et `write()`, limitant ainsi les copies des tableaux.

Cette implémentation remplace désormais l'ancienne version TCP utilisant JNI, dans la mesure où elle lui est strictement équivalente.

3.2 Communications Multi-Grille

L'architecture réseau, du cas d'utilisation qui nous intéresse, dispose d'une grille expérimentale et d'une grappe locale, chacune accessible par une passerelle via le protocole ssh sur le port 22 (cf figure 3.1 page suivante) :

- Grid'5000 (acces.sophia.grid5000.fr)
- Les eons (ssh-sop.inria.fr)

Chaque nœud peut accéder librement au réseau Internet. L'accès ssh se fait avec authentification par clé RSA [RIV 78].

Les deux architecture réseaux ne permettent pas de communication directe de l'une à l'autre, il faut faire transiter les requêtes par leur passerelle respective. Les protocoles de communications de ProActive ne permettent pas ce cas d'utilisation.

3.2.1 Proxy

Le protocole rmi+ssh suppose une connexion utilisant le protocole ssh possible entre tous les noeuds. Nous nous sommes inspiré d'une option de configuration du client ssh OpenSSH [3], la « ProxyCommand ». La « ProxyCommand » est une commande exécutée sur une machine passerelle avant d'exécuter la commande désirée. Elle permet de simuler le comportement d'un serveur mandataire (`proxy`) sur la passerelle, et donc de contacter la machine derrière la passerelle de façon transparente. Récapitulatif des exigences du serveur mandataire dans notre cas, le serveur mandataire doit :

- Retransmettre les requêtes qu'il reçoit vers leur destinataire véritable
- Agir de façon transparente
- Garantir la sécurité des informations transitant sur le réseau Internet

[3] `http://www.openssh.com/`

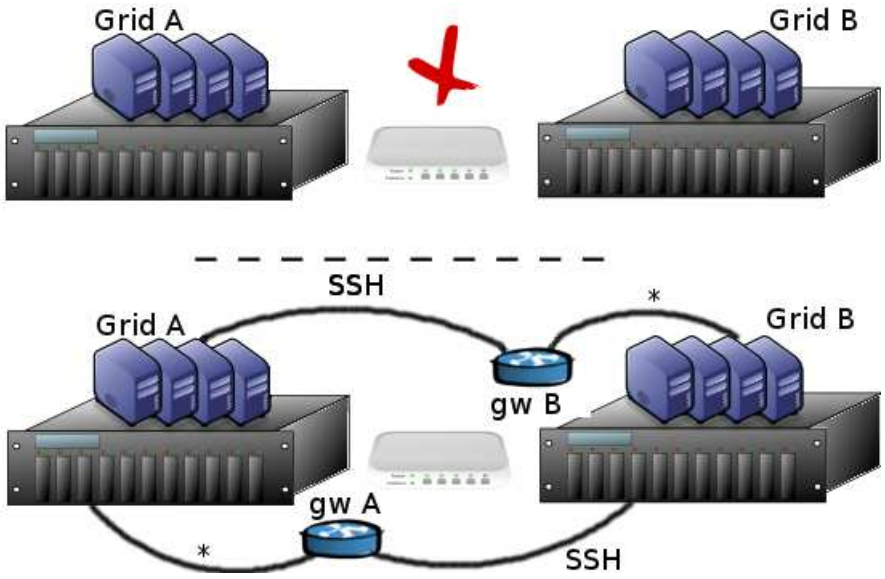

Fig. 3.1 – Architecture réseau de deux grilles et leurs passerelles (gw)

Dès lors il paraît intéressant d'implémenter la « ProxyCommand », car le protocole SSH, garantit la sécurité des informations échangées et les passerelles avec lesquelles nous travaillons fournissent un service SSH ce qui nous évite de nécessiter des droits d'administration sur les passerelles (ouverture de port), ou bien d'implémenter et d'exécuter un serveur mandataire sur la passerelle (cf. Fig. 3.2 page suivante).

Principes :

Haut Niveau Le proxy fonctionne en trois étapes, d'abord un serveur mandataire est démarré sur la passerelle et écoute sur un port précis (33647 par défaut). Ensuite la requête RMI est interceptée et redirigée sur la passerelle, sur le port en écoute. Pour finir le serveur mandataire transmet la requête à la machine véritablement ciblée sur le port initialement indiqué.

Bas Niveau Le serveur mandataire sera donc simulé par l'exécution d'une commande sur le serveur SSH. La commande `netcat` sera :

```
> nc hote_désiré port_désiré
```

ce qui aura pour conséquence d'ouvrir une connexion TCP entre la passerelle et l'objet distant (ou registre RMI contenant la référence vers l'objet distant) sur l'hôte distant. Il suffit alors de connecter les flux d'entrée et de sortie du protocole RMI et ceux de la commande SSH précédemment exécutée.

3.2.2 Implémentation de la « ProxyCommand »

Le choix de conception qui me paraissait le plus pertinent est de concevoir la « ProxyCommand » comme une extension du protocole RMI+SSH de ProActive (cf. Fig. 3.3 page 20).

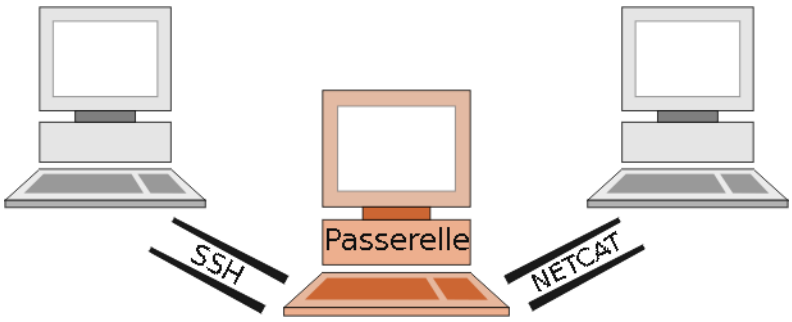

FIG. 3.2 – Illustration de la Proxy Command SSH

La classe `UnicastRemoteObject` utilisée par RMI requiert une socket pour communiquer avec l'objet distant/registre RMI, nous avons donc écrit la classe `SshProxy` qui étend la classe `java.net.Socket` dont le travail est d'exécuter la « ProxyCommand » sur la passerelle et de fournir les flux d'entrées et de sorties à RMI.

3.2.3 Utilisation de la « ProxyCommand »

Les options de configuration de la « ProxyCommand » sont lues de la même façon que les options de configuration de ssh (cf. 14 page 11). La propriété

`proactive.communication.protocol`

devra avoir pour valeur `rmissh` (puisque la « ProxyCommand » n'est qu'une extension du protocole RMI+SSH). La propriété :

`proactive.ssh.proxy.gateway`

contient les informations de correspondance entre machines et passerelles. Sa syntaxe est la suivante :

`machine1:passerelle1:port1;machine2:passerelle2:port2;...`

Il est aussi possible de définir un ensemble de machines de la façon suivante :

`*.domaine:passerelle:port;...`

Les passerelles étant accessibles directement peuvent aussi être spécifiées ainsi :

`*.domaine:passerelle:port;passerelle:none`

Si le nom d'une machine ne correspond à aucune des options de configuration spécifiées, (ou bien dont la passerelle est `none`), alors le protocole RMI+SSH est utilisé.

3.2.4 Avantages et inconvénients

Avantages L'utilisation de la « ProxyCommand » est complètement transparente pour le protocole RMI+SSH. Les tests effectués lorsque la machine distante ne la nécessite pas, sont

négligeables. De plus, aucune installation ou modification de l'architecture de la grille/passerelle n'est nécessaire, et toute la configuration se fait à l'aide des propriétés précédemment citées.

Nous allons donc pouvoir utiliser plusieurs ressources hétérogènes (grilles, grappes, « cloud »[4] (ex : Amazon ec2)) pour exécuter une même application, ce qui était impossible auparavant.

Inconvénients Un goulot d'étranglement est alors créé sur la passerelle, mais ceci est dû à l'architecture réseau de la grille qui l'impose pour des raisons de sécurité.

[4] `http://fr.wikipedia.org/wiki/Cloud_computing`

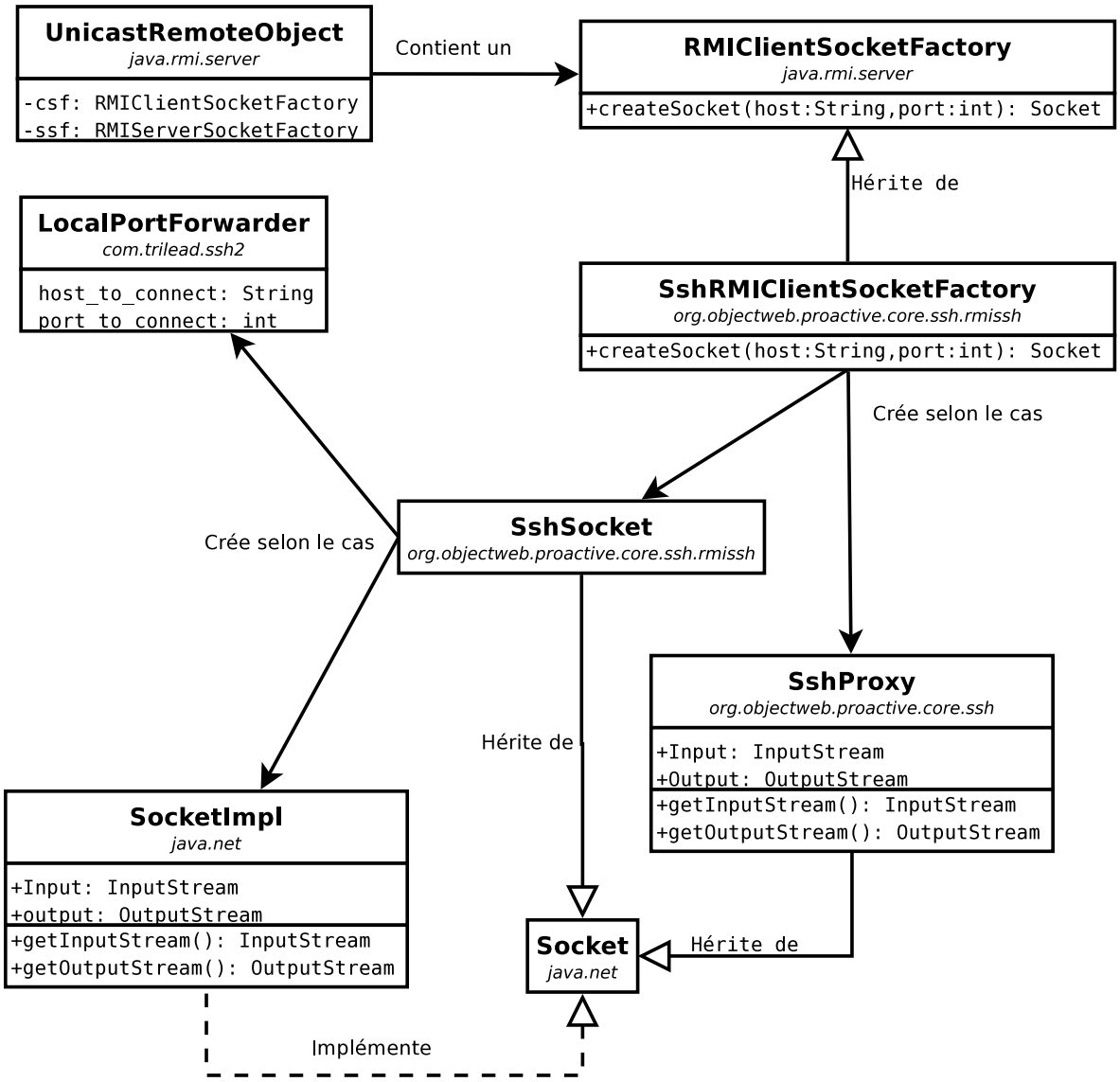

FIG. 3.3 – Diagramme UML de la Proxy Command SSH

Chapitre 4

Gestion de projet

Nous avons utilisés différents services et outils dans le cadre de notre TER.

4.1 Services

Centre de documentation

Le centre de documentation Descartes de l'INRIA nous a prêté différents ouvrages, notamment sur Unix [STE 90] [COM 01]. Le réseau de documentation nous a permis d'obtenir des ouvrages concernant JAVA NIO [HIT 02], et JNI [LIA 99], [GOR 98]. Ces trois derniers livres ayant été commandés auprès du centre de recherche de l'INRIA à Grenoble.

INRIAGforge

Le site `http://gforge.inria.fr/` propose aux membres de l'INRIA d'héberger les projets publics et privés des équipes de recherche du centre. Il permet notamment de faciliter le développement de projets entre différents sites de l'INRIA par exemple, ou avec des sociétés partenaires. Les projets PROACTIVE et DISCOGRID sont tous deux hébergés sur cette plateforme de développement.

Pour le besoin du TER, Elton MATHIAS a créé deux branches dans le dépot de DISCOGRID. Une nommée « dg-deployment » pour Anaud, et « dg-communication » pour Cyprien.

Internet Relay Chat

Les membres de l'équipe OASIS et d'ACTIVEEON se rejoignent sur un canal de discussion utilisant le protocole IRC. Ce biais permet de dialoguer directement aux développeurs de PROACTIVE, et donc d'obtenir rapidement des informations sur un problème par exemple.

4.2 Outils

OAR

OAR est un gestionnaire de ressource pour grilles étendues développé par l'INRIA, et hébergé par Gforge. Son principal usage est de réserver des machines sur une grille, ainsi que de lancer des scripts de traitement par lot (batch) sur ces machines.

Voici un résumé des principales commandes dont nous nous sommes servis :

- `oarstat` qui permet de voir les réservations en cours.
- `oarsub` qui permet de soumettre une nouvelle réservation. Peuvent être spécifié, le ou les différente(s) grappe(s) à utiliser, le temps de la réservation, sa date de début, le nombre de noeud/processeur/coeur à réserver, ...
- `oardel` qui permet de supprimer une réservation
- `oarsh` qui permet d'ouvrir un shell sur la/les machine(s) précédemment réservées.

4.2.1 Travail collaboratif

Subversion

Les projets sont accessibles à travers l'outil de gestion de version « subversion [1] »

Git

C'est un autre système de gestion de version, développé par Linus Torvalds, il possède quelques différences avec subversion. Nous nous en sommes servis pour l'édition des documents écrits avec LaTeX justement.

GIMP et dia

Nous nous sommes servis du logiciel GNU Image Manipulation Program (GIMP) pour l'édition d'images, logos, et schémas. Le logiciel dia [2] fut utilisé pour la réalisation des diagrammes UML.

4.2.2 Environnement de développement

Nous avons utilisé plusieurs environnements de développement, en fonction du langage utilisé.

Eclipse

Il a servi principalement à l'édition du code Java, et XML. Ses outils permettent également de vérifier au vol la validité et la correction du code Java, et de tester certains bouts de code. L'édition de fichier XML a été plutôt rare, pour modifier les scripts de compilation utilisé par Ant notamment.

Emacs

Inutile de le présenter, mais pour ceux qui ne connaîtraient pas ce puissant outil développé à l'origine par R. M. Stallman, la documentation est accessible sur la page du logiciel à l'adresse `http://www.gnu.org/software/emacs/`.

[1] `http://subversion.tigris.org/`
[2] `http://live.gnome.org/Dia`

4.2.3 Divers

LaTeX

Ce logiciel nous a permis l'édition de documents au format PDF, notamment ce rapport, ainsi que les transparents utilisés lors des présentations, à l'aide de l'extension Beamer.

Chapitre 5

Conclusion

5.1 Ce qui a été fait

Au niveau des communications internes, nous avons proposée une alternative utilisant une approche différente, mais en même temps plus portable et sensée être plus efficace.

Pour ce qui est des communications externes, la « Proxy Command » a été implémentée dans le but de permettre, de façon totalement transparente, les communications entre des machines situées derrières des passerelles ; ce qui correspond au cas réel de GRID'5000.

5.2 Problèmes rencontrés

L'implémentation SSH GANYMED de Trilead venait avec peu de documentation, et son développement à été abandonné par cette société courant mai.

5.3 Ce qu'il reste à faire

Pour les communications internes, il faudrait mesurer le gain de performance effectif entre la version IPC et celle TCP avec JAVA NIO, sur différentes configurations de déploiement. Il y a plusieurs points où le code pourrait être simplifié, rendu plus robuste, notamment en cas d'erreurs d'entrée et/ou de sortie sur les sockets, afin de renforcer la robustesse de l'implémentation.

En ce qui concerne les communications externes, il faudrait tester intensivement l'implémentation proposée, afin de mesurer le coût lié au chiffrement des données fait par le protocole SSH au niveau du serveur mandataire. La gestion d'erreurs est aussi à améliorer, afin d'éviter la levée de certaines exceptions qui pourraient éventuellement être fatales et terminer l'application.

Bibliographie

[BAD 06] Laurent Baduel, Françoise Baude, Denis Caromel, Arnaud Contes, Fabrice Huet, Matthieu Morel, and Romain Quilici. " *Grid Computing : Software Environments and Tools*", chapter Programming, Deploying, Composing, for the Grid. Springer-Verlag, January 2006.

[BAU 08] Francoise Baude, Denis Caromel, Cédric Dalmasso, Marco Danelutto, Vladimir Getov, Ludovic Henrio, and Christian Pérez. GCM : A Grid Extension to Fractal for Autonomous Distributed Components. *Annals of Telecommunications*, 2008.

[CAR 99] Denis Caromel, and Isabelle Attali. Oasis resarch team, INRIA sophia antipolis, 1999. `http://www-sop.inria.fr/oasis/`.

[CAR 05] Denis Caromel, and Ludovic Henrio. " *A Theory of Distributed Object*". Springer-Verlag, 2005.

[CAR 07] Denis Caromel. ActiveEon, 2007. `http://www.activeeon.com/`.

[COM 01] Douglas E. Comer, and David L. Stevens. " *Internetworking with TCP/IP, Vol. 3 : Client-Server Programming and Applications, Linux/Posix Sockets Version*". Prentice-Hall, 2001.

[DIS 05] DiscoGrid. Agence national de la recherche, 2005. `http://www-sop.inria.fr/nachos/team_members/Stephane.Lanteri/DiscoGrid/`.

[GOR 98] Rob Gordon. " *Essential JNI : Java Native Interface*". Prentice-Hall, 1998.

[GRI 05] Grid5000. INRIA, CNRS, 2005. `https://www.grid5000.fr`.

[HIT 02] Ron Hitchens. " *Java NIO*". O'Reilly & Associates, Inc., 2002.

[LIA 99] Sheng Liang. " *Java Native Interface : Programmer's Guide and Specification*". Addison-Wesley, 1999.

[PRO 99] ProActive. INRIA, 1999. `http://proactive.inria.fr/`.

[PUG 03] William Pugh, and Jaime Spacco. " Mpjava : High-performance message passing in java using java.nio". In *Proceedings of the Workshop on Languages and Compilers for Parallel Computing (LCPC'03), College Station, TX*, pages 323–339, October 2003.

[RIV 78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2) :120–126, 1978.

[STE 90] Richard W. Stevens. " *UNIX Network Programming*". Software Series. Prentice-Hall, 1990.

[YLO 06] T. Ylonen, and C. Lonvick. The Secure Shell (SSH) Protocol Architecture. RFC 4251 (Proposed Standard), January 2006.