GoDIET: Un outil pour le déploiement de DIET

Eddy Caron, Holly Dail

▶ **To cite this version:**

Eddy Caron, Holly Dail. GoDIET: Un outil pour le déploiement de DIET. [Research Report] LIP RR-2004-49, Laboratoire de l'informatique du parallélisme. 2004, 2+8p. hal-02102039

HAL Id: hal-02102039

https://hal-lara.archives-ouvertes.fr/hal-02102039

Submitted on 17 Apr 2019

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n° 5668

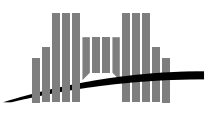
GoDIET: Un outil pour le déploiement de DIET

Eddy Caron ,
Holly Dail

Novembre 2004

Research Report N° 2004-49

École Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : +33(0)4.72.72.80.37
Télécopieur : +33(0)4.72.72.80.80
Adresse électronique : `lip@ens-lyon.fr`



GoDIET: Un outil pour le déploiement de DIET

Eddy Caron , Holly Dail

Novembre 2004

Abstract

In this article, we address the automatic configuration, launch, and management of a distributed middleware platform for the Grid of type Application Service Provider (ASP) called Distributed Interactive Engineering Toolbox (DIET). The difficulty of this task is due to the architecture of DIET, which is distributed and hierarchical. The need for an automated tool for these tasks is reinforced by the diversity and number of components for the DIET middleware. In this article, we present GoDIET, a new tool created to satisfy these needs, though the solutions presented are valid for any distributed, hierarchical environment. The principles underlying GoDIET will be detailed by example with the configuration and launch of DIET and LogService, an external service for the management of traces for distributed software components. We then present a series of experiments that permit the evaluation of the performance and efficacity of GoDIET. We also evaluate the robustness of the DIET platform for a large number of servers.

Keywords: Deployment, PSE, Grid computing

Résumé

Dans cet article, nous exposons les travaux menés autour de la configuration, du lancement et de la gestion de DIET, un intergiciel de type ASP (*Application Service Provider*) pour la grille. La difficulté de cette tâche repose sur l'architecture de DIET qui est distribuée et hiérarchique. Le besoin de disposer de ce type d'outils est renforcé par la diversité et le nombre des éléments de cet intergiciel. Dans cet article, nous présenterons GoDIET, un nouvel outil adapté aux contraintes de DIET, cependant les concepts mis en œuvre restent valides pour tout environnement distribué et hiérarchique. Le principe de fonctionnement de GoDIET sera détaillé au travers de son application pour la gestion de DIET et du LogService, un service externe pour la gestion de traces d'éléments distribués. Enfin, nous présentons une série d'expérimentations qui permettent d'évaluer la performance et l'efficacité de GoDIET. Dans cette même série d'expériences la robustesse de la plate-forme DIET sur un grand nombre de serveurs sera également étudiée.

Mots-clés: Déploiement, PSE, Calcul sur la grille

1 Introduction

Dans cet article, nous allons présenter les travaux menés nous permettant de déployer sur la grille une plate-forme de type ASP (Application Service Provider) appelée DIET (Distributed Interactive Engineering Toolbox). Nous nous sommes intéressés ici à automatiser la configuration, le déploiement, et la gestion d'une plate-forme distribuée composée par des agents et des serveurs. Dans cet article, l'ensemble de ces tâches seront associées au terme "déploiement", même si cela ne prend pas en considération le déploiement des applications en elles-mêmes. On parle de déploiement de l'intergiciel et non de déploiement d'applications.

Afin d'assurer une bonne extensibilité et en respectant les contraintes physiques des réseaux, l'architecture de DIET repose sur un ensemble d'éléments structurés de manière hiérarchique. Dans ce cadre de plate-forme distribuée, il est important de pouvoir disposer d'un outil facilitant une mise en place complexe due à la diversité et au nombre des éléments. L'état de l'art présenté en Section 2 vient illustrer ce propos, et souligne un manque vis à vis d'outils de déploiement génériques facilitant leur utilisation dans un autre cadre que celui pour lequel ils ont été conçu. Il était donc difficile d'adapter l'existant à notre intergiciel. Après une rapide introduction de DIET et de ses éléments en Section 3, nous présenterons l'outil de déploiement GoDIET en Section 4. Nous évoquerons également les fonctionnalités de base pour la gestion de la plate-forme. Le principe de fonctionnement de GoDIET sera détaillé au travers d'un exemple de mise en œuvre du déploiement de DIET et du LogService, un service externe pour la gestion de traces pour les éléments logiciel distribués. En Section 5 cet outil est évalué par une série d'expérimentations menées sur un réseau à haut débit, VTHD. Ces expériences permettrons également de valider la robustesse de notre plate-forme de calcul sur la grille sur un grand nombre de serveurs.

2 L'état de l'art

JXTA Distributed Framework (JDF) [1] a été conçu afin de faciliter les tests sur les systèmes basés sur JXTA. JDF utilise un fichier XML pour la description du réseau logique de pairs JXTA et des ressources machines utilisées. JDF est un ensemble de scripts de commandes (*scripts shell*) qui nécessite une machine virtuelle Java pour la configuration des pairs. Le déploiement est réalisé par ssh ou rsh. Dans l'optique d'intégrer un service de partage de données distribuées pair-à-pair appelé JuxMem, basé sur JXTA, nous travaillons à réaliser une extension de GoDIET qui permettra un déploiement commun des deux plates-formes.

APST (AppLeS Parameter Sweep Template) [4] offre un déploiement à large échelle d'applications basées sur le modèle de balayage de paramètres (*parameter sweep*). Les ressources, les tâches et les paramètres sont définis dans un fichier XML. APST se charge ensuite de gérer la distribution des fichiers, l'ordonnancement des tâches et de récupérer le résultat.

Nous avons brièvement cité ici deux exemples d'outils fournissant des mécanismes, mais bien que les projets de déploiement d'outils pour la grille soient nombreux, aucun à notre connaissance ne fournit un support pour l'intégration d'une application (ou d'un intergiciel) spécifique. Par exemple, dans notre cas nous cherchions un outil capable de gérer un déploiement d'entités distribuées et hiérarchiques (ce qui signifie avec des dépendances). L'outil de déploiement que nous proposons dans cet article tente de répondre à ces contraintes. À plus long terme nous espérons orienter cet outil vers un outil plus générique et facilement transposable à d'autres intergiciels.

3 Présentation de DIET

DIET [3], est un ensemble hiérarchique d'entités pour l'élaboration d'applications basées sur des serveurs de calcul. L'objectif de cet intergiciel est de fournir une interface suffisamment simple pour masquer aux applications l'infrastructure distribuée. L'environnement proposé est capable de déterminer le serveur approprié en tenant compte de la requête de calcul, mais également de la localisation des données (qui peuvent être n'importe où sur le système suite à une précédente exécution, par exemple) ou encore des ressources disponibles à cet instant. L'architecture de DIET

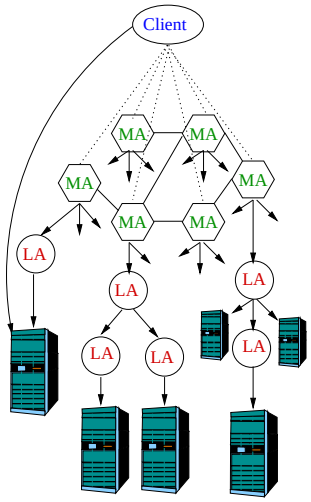


Figure 1: Vue générale de DIET.

a été pensée selon une approche hiérarchique afin d'assurer une meilleure extensibilité et de tenir compte des contraintes de déploiement du réseau physique. L'architecture de DIET est représentée Figure 1. DIET repose sur les éléments suivants :

Client Lorsque l'on évoque le client, il s'agit du client applicatif qui utilise DIET pour résoudre des problèmes. Plusieurs clients peuvent se connecter à DIET. Un problème peut être soumis de manière synchrone (le client reste en attente du résultat) ou de manière asynchrone (le contrôle est rendu au programme après la soumission). Des problèmes peuvent être soumis depuis un site web, un PSE (Problem Solving Environment) comme Scilab [2], ou d'un programme compilé.

Master Agent (MA) Un MA est directement relié aux clients. Il reçoit des requêtes de calcul des clients et choisit un (ou plusieurs) SeD qui sont capables de résoudre le problème en un temps raisonnable. Un MA possède les mêmes informations qu'un LA, mais il a une vue globale (et de haut niveau) de tous les problèmes qui peuvent être résolus et de toutes les données qui sont distribuées dans tous ses sous-arbres afin d'accélérer la recherche d'un SeD capable de résoudre une requête.

Local Agent (LA) Un LA compose un niveau hiérarchique dans les agents DIET. Il peut être le lien entre un MA et un SeD, deux LA ou un LA et un SeD. Son but est de diffuser les requêtes et les informations entre les MAs et les SeDs. Il tient à jour une liste des requêtes en cours de traitement et, pour chacun de ses sous-arbres, le nombre de serveurs pouvant résoudre un problème donné, ainsi que des informations à propos des données.

Server Daemon (SeD) Un SeD est le point d'entrée d'un serveur de calcul. Il gère l'ensemble des ressources qui lui sont allouées. Il peut s'agir d'un processeur, comme d'un cluster (dans le cas d'applications parallèles). Il tient à jour une liste des données disponibles sur un serveur (éventuellement avec leur distribution et le moyen d'y accéder), une liste des problèmes qui peuvent y être résolus, et toutes les informations concernant sa charge (charge CPU, mémoire disponible, taille du disque, etc.). Pour réaliser un meilleur ordonnancement des requêtes, un SeD peut utiliser FAST [5], un outil de modélisation des performances dynamiques.

En résumé, le client DIET soumet les requêtes aux ordonnanceurs (appelés agents), récupère la référence du serveur choisi par les agents, enfin, il communique les données à ce serveur afin d'effectuer le calcul.

4 Architecture de GoDIET

L'objectif de GoDIET est de fournir un outil facilitant le déploiement de plates-formes DIET et des services associés sur la grille. Les caractéristiques visées par cet outil sont la portabilité, l'extensibilité, une interface en mode console et une interface graphique. Autant de contraintes qui ont été prises en compte afin de définir notre prototype.

Pour assurer la portabilité de GoDIET, le prototype est implémenté en Java. Cela favorise l'interface avec des outils graphiques basés sur JAVA et facilite le prototypage de l'application. La description des ressources de la plate-forme et des logiciels à déployer sont décrits par un fichier XML. Afin de valider les instances du fichier XML, nous avons défini un fichier DTD (Document Type Definition) associé à un analyseur syntaxique (*paser*) de vérification. Un fichier XML de GoDIET inclut la description des éléments et de leur hiérarchisation, des machines physiques, des espaces disques où seront déposés les éléments et leurs données, la localisation des binaires pour le lancement des éléments, etc. La séparation de la définition des ressources et de la définition de la hiérarchie permet de partitionner la configuration. Il n'est pas nécessaire de ré-écrire la description de la plate-forme lorsque l'on souhaite effectuer un nouveau déploiement en utilisant les mêmes ressources et réciproquement.

L'interface utilisateur de base est disponible en mode console, et peut être utilisée sur n'importe quelle machine disposant de Java et ne nécessite pas d'avoir accès à une interface graphique. De plus, un mode automatique permet de déployer, de tester et de stopper la plate-forme selon un traitement par lots (batch). De plus la console offre des fonctionnalités telles qu'un déploiement conforme à la description du fichier XML, l'affichage du statut des nœuds (Exemple: `machine01` est un LA appelé `LA1`) ou encore la possibilité de stopper la plate-forme.

Nous utilisons les commandes `scp` et `ssh` pour effectuer des transferts de fichiers sécurisés et exécuter les tâches. `ssh` est un outil d'accès distant fortement répandu et universel. Son utilisation est donc plus que familière aux utilisateurs de la grille. Avec une commande `ssh` étendue, GoDIET est capable de fournir les variables d'environnement et les paramètres d'un processus, de sauvegarder la sortie standard et la sortie d'erreur de chaque processus en vue de corriger les erreurs, et de fournir le PID du processus chargé (qui pourra être utilisé pour la gestion de la plate-forme). Afin d'illustrer nos propos, voici un exemple de ce type de commande:

```
/bin/sh -c ( /bin/echo ≻≺
"export PATH=/home/user/local/bin/:$PATH ; ≻≺
export LD_LIBRARY_PATH=/home/user/local/lib ; ≻≺
export OMNIORB_CONFIG=/home/user/godiet_s/run_04Jul01/omniORB4.cfg; ≻≺
cd /home/user/godiet_s/run_04Jul01; ≻≺
nohup dietAgent ./MA_0.cfg < /dev/null > MA_0.out 2> MA_0.err &" ; ≻≺
/bin/echo '/bin/echo ${!}' )  ≻≺
| /usr/bin/ssh -q user@ls2.ens.vthd.prd.fr /bin/sh -
```

Avec cette commande, on lance un binaire sur une machine à distance avec un `PATH` et un `LD_LIBRARY_PATH` configurable selon les logiciels que l'on souhaite tester et on garde les sorties `stdout` et `stderr` de l'application dans des fichiers séparés (log et erreur). On peut récupérer le `PID` (process identifier) qui peut-être utilisé par GoDIET pour arrêter le processus plus tard.

Afin d'enchaîner les différents lancements, il est nécessaire de pouvoir exécuter une commande à distance (via ssh) non bloquante. Pour cela nous utilisons la commande `nohup` qui indique au processus qu'il peut continuer même si la connexion est perdue. Par ailleurs nous devons éviter les attentes provenant des flux `stdin`, `stdout` et `stderr`. Afin de ne pas surcharger la machine GoDIET et surtout éviter les limitations provoquées par un nombre de sessions ssh ouvertes, on n'effectue qu'une connexion ssh à la fois. Nous avons effectué arbitrairement des tests jusqu'à 350 lancements séquentiels les uns à la suite des autres, mais l'aspect séquentiel de ce traitement ne pose aucune contrainte.

Dans la section suivante nous nous intéresserons aux aspects de déploiement et à la gestion du LogService utilisant GoDIET. Afin d'illustrer ces propos, la Figure 2 présente une vue d'ensemble de l'intégration de GoDIET et des différents services liés à DIET. Parmi ces services, citons

également l'outil de visualisation appelé VizDIET. Nous ne traiterons pas cet outil qui sort du cadre qui nous intéresse ici, mais signalons que ce logiciel compte intégrer les développements de GoDIET afin de coupler visualisation et administration de la plate-forme.

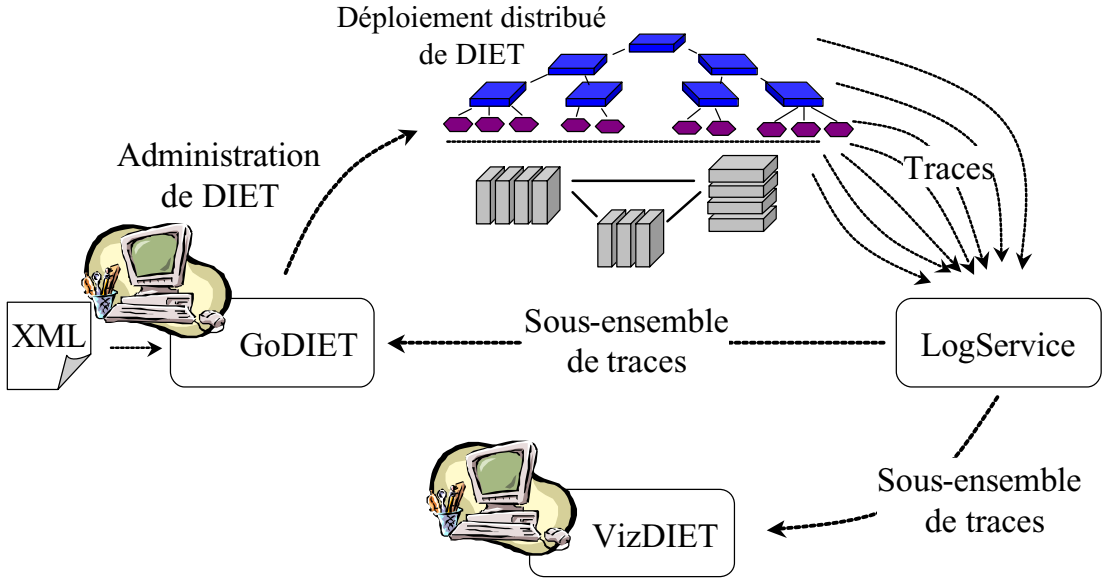


Figure 2: Interaction des différents services de la plate-forme DIET

4.1 Déploiement de DIET

La première tâche pour GoDIET est le déploiement de DIET sur la grille. La plate-forme est construite selon une hiérarchie, chaque élément étant connecté à son père. L'architecture de la hiérarchie est décrite par GoDIET au travers des fichiers de configuration de chacun des éléments. En effet, chaque élément a dans son fichier le nom de son parent (exception faite pour le MA, racine de la hiérarchie). DIET est écrit en CORBA/C++ et utilise un service de nommage centralisé (OmniNames) qui permet aux éléments déployés de s'identifier avec une simple chaîne de caractère. GoDIET fournit la localisation de ce service de nommage et le port correspondant en se basant sur le fichier de configuration de chaque élément. Plusieurs déploiements de DIET peuvent être effectués sur les mêmes machines sans conflit tant que les différents services de nommage alors activés, utilisent des ports différents.

Un certain nombre de facteurs relatifs aux machines cibles sont également incorporés dans le fichier de configuration. Pour les plates-formes sans résolution de nom (DNS non disponible) ou pour les machines disposant de plusieurs interfaces réseaux, les éléments peuvent être assignés manuellement à un hôte final (*endpoint hostname*) ou à l'IP définie dans leur fichier de configuration. En utilisant le `hostname`, les éléments peuvent être détectés correctement par le service de nommage même dans un environnement réseau difficile (prise en compte des NAT). De manière identique, un port `endpoint` peut être défini afin de prévoir les cas où l'accès au port est limité par les règles du pare-feu. Toutes ces fonctionnalités peuvent être facilement activées via le fichier XML. GoDIET ajoutera ces fonctionnalités à chaque élément lors de la phase de déploiement. DIET offre le dispositif et la flexibilité nécessaire afin de prendre en compte la grande variété de déploiement, GoDIET doit être alors capable de gérer chacune des fonctionnalités de DIET.

La Figure 3 décrit chaque étape de la mise en œuvre d'une architecture DIET simple. Après avoir chargé le service de nommage, le MA est le premier élément démarré (Figure 3 (1)). Il est alors en attente de connexion d'un agent ou de requêtes en provenance des clients. Lors de son initialisation le LA vient s'inscrire auprès du MA (Figure 3 (2)).

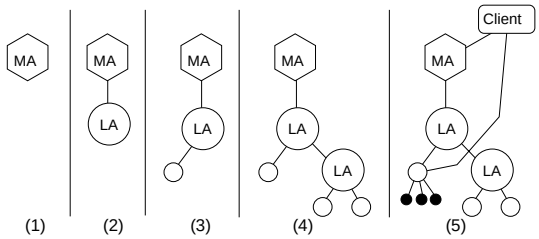


Figure 3: Initialisation d'une plate-forme DIET .

Deux types d'éléments peuvent alors être connectés au LA: soit un SeD (Figure 3 (3)) soit un autre LA afin d'ajouter un niveau de hiérarchie à la branche (Figure 3 (4)). Le client peut alors se connecter au MA afin de soumettre une requête (Figure 3 (5)).

Un point difficile pour un déploiement correct de DIET est que chaque élément est déployé par un appel ssh rapide qui s'exécute sans connaissance de l'initialisation des autres éléments. Par conséquent, il est important d'assurer une cohérence dans la déclaration auprès du service de nommage afin de respecter la hiérarchie. En effet, un élément ne peut se connecter à son père que si ce dernier est lui-même lancé et par conséquent connu du service de nommage. Dans un réseau peu hétérogène, l'utilisation d'une fonction d'attente (ie. `sleep`) est suffisante pour gérer les dépendances. Dans le cas d'un environnement de type grille ou les réseaux sont hétérogènes et dynamiques, une approche adaptative est nécessaire. Nous présentons une méthode de ce type dans la section suivante.

4.2 Déploiement de LogService

LogService est un logiciel de génération de fichiers de traces (appelé aussi fichier de logs) développé pour un environnement distribué. Ce logiciel permet de suivre, par échange de messages (ou de fichiers), les traces d'exécution des différents agents DIET.

Notons que le LogService a été conçu afin de transmettre des traces à un ensemble d'éléments en "écoute" de la plate-forme. Des mécanismes de filtre peuvent être mis en place. Notons également que la panne d'un élément DIET n'a pas d'implication sur la stabilité du LogService.

LogService [6] est composé de deux modules : le LogCentral et le LogManager. Le LogManager est situé sur chaque agent et permet de collecter les informations et de les envoyer au LogCentral. Celui-ci collecte tous les messages des LogManagers et met à jour un fichier de logs. Ce logiciel permet une synchronisation des messages de traces entre les agents et le LogCentral. En effet, dans un environnement distribué, il peut y avoir des horloges différentes entre les différents agents et la machine sur laquelle tourne LogCentral.

Le déploiement de LogService peut être facilement réalisé avec GoDIET. LogService doit être lancé une fois l'omniNames activé (dans le cas d'une utilisation avec DIET, ce dernier correspond à celui utilisé par DIET et décrit dans les fichiers de configuration correspondants). La procédure de déploiement est réentrante dans le sens où GoDIET est utilisé afin de déployer LogService, puis GoDIET utilise les informations fournies par LogService pour avoir les informations sur l'état de la plate-forme.

Un thread distinct est en attente des messages qui arrivent, et alerte le thread de chargement lorsqu'un élément est enregistré auprès du LogService. Le thread de chargement utilise alors le retour en temps réel de LogService pour assurer le chargement des éléments de DIET en respectant leurs dépendances. Ainsi un élément ne va être chargé que si son père a été au préalable chargé. Cette approche adaptative est adaptée aux environnements réseaux hétérogènes. En effet, on garantit le respect des dépendances sans avoir à considérer la latence (variant en fonction du type réseau) lors de l'enregistrement d'un élément au LogService.

5 Expérimentations

Nous présentons une série d'expérimentations afin d'évaluer les performances et l'efficacité de GoDIET pour le déploiement. Ces expérimentations permettront également de valider la robustesse de la plate-forme DIET sur un grand nombre de serveurs.

Notre approche expérimentale est la suivante, nous avons sélectionné une plate-forme expérimentale sur la grille ainsi qu'une hiérarchie DIET adaptée à cette plate-forme physique. Nous avons défini le fichier XML correspondant et mesuré le temps pris par GoDIET pour charger la plate-forme logicielle, effectuer une série de tests afin de valider la plate-forme déployée et désinstaller le tout en utilisant GoDIET.

Un aspect clef de ce processus est de coordonner le chargement des éléments en respectant les dépendances des éléments. Afin de construire correctement le déploiement de DIET, chaque agent doit s'enregistrer auprès du service de nommage avant qu'un des éléments qui dépend de lui ne tente de le contacter. Les serveurs n'ont pas besoin de se coordonner entre eux, il leur suffit juste que l'agent à qui ils doivent se rattacher soit chargé. Nous avons testé trois approches différentes qui prennent en compte la gestion des dépendances entre les éléments:

Temps Fixe: C'est l'approche la plus simple, elle implique simplement d'attendre un certain temps (fixé) avant de charger un élément dépendant de celui qui vient d'être lancé (et qui est en train de s'identifier auprès du service de nommage). Nos expériences sont réalisées avec 3 secondes d'attente après le lancement de omniNames et LogCentral, 2 secondes après chaque agent et une seconde après chaque serveur. Ces temps ont été déterminés expérimentalement sur un petit ensemble d'exécutions.

Avec Retour: Cette approche utilise les retours en temps réel du LogService afin de gérer le processus de chargement des éléments. GoDIET attend qu'un élément soit enregistré auprès du LogService avant de charger les éléments qui en dépendent.

Mixte: Cette approche utilise la méthode "avec retour" pour le chargement d'omniNames, de LogCentral et des agents DIET mais n'introduit pas de pause entre le chargement des serveurs.

Pour une plate-forme avec S serveurs, nous exécutons le programme client avec S appels séquentiels. Rappelons que les clients utilisent un processus en deux phases pour interagir avec DIET: il envoie une requête afin de trouver le serveur approprié dans la hiérarchie (phase d'ordonnancement), et ensuite le calcul sera transmis directement au serveur pour réaliser le calcul (phase de calcul). Dans nos expériences, nous avons considéré que si nous avons S réponses valides de la hiérarchie d'agent, la hiérarchie d'agent fonctionne comme attendu. Nous devons ensuite nous assurer que chacune des S requêtes est assignée à un unique serveur (selon un ordonnancement tourniquet (*round-robin*)). Si la phase de calcul est réalisée pour les S requêtes, nous considérons que tous les serveurs fonctionnent correctement. Enfin, pour chaque plate-forme nous mesurons le temps total pour l'exécution de ces S requêtes et divisons cela par le nombre de requêtes pour trouver la latence moyenne de DIET (ordonnancement et calcul).

Pour les expériences, nous utilisons une grille de quatre sites avec 102 machines et 342 processeurs au total; le tableau 1 donne plus d'informations sur cette plate-forme. Cette grille à grande échelle est basée sur un réseau à haut débit, VTHD (Vraiment Très Haut Débit)

5.1 Expérimentations pour l'évaluation des performances de lancement de la plate-forme

Pour ces expérimentations, le type de hiérarchie de la plate-forme DIET est fixe et nous faisons varier le nombre de serveurs. La hiérarchie est composée d'un MA sur la grappe `ls`, un LA sur chacun des quatre autres sites, avec entre 2 et 32 serveurs sur chaque site (ce qui nous donne entre 8 et 128 serveurs au total). Les temps de lancement obtenus pour ces différentes configurations sont présentées Figure 4. Chaque expérience a été menée trois fois pour chaque plate-forme.

Grappe	Site	Nœuds	Nb Procs	Type	Mem.
PARACI	Rennes	64	4	Xeon 2.4 GHz	1 GB
LS	Lyon	16	2	Pentium III 1.4 GHz	1 GB
Cristal	Rocquencourt	13	2	Xeon 2.8 GHz	2 GB
Cobalt	Grenoble	9	2	Pentium III 1.4 GHz	1 GB

Table 1: Description de la plate-forme d'expérimentation. Le nombre de processeurs et la quantité mémoire sont donnés par nœud.

Le temps de chargement est évidemment dépendant du nombre de serveurs que l'on souhaite charger. Le coût de l'utilisation d'un temps d'attente est significatif mais en contrepartie offre la meilleure stabilité pour le lancement de la plate-forme. En utilisant, les retours des traces du LogService pour l'ensemble des éléments nous avons une amélioration cependant le temps de chargement varie en fonction du réseau de communication sous-jacent. La méthode la plus rapide est la méthode mixte, qui n'utilise le retour de traces que pour les agents.

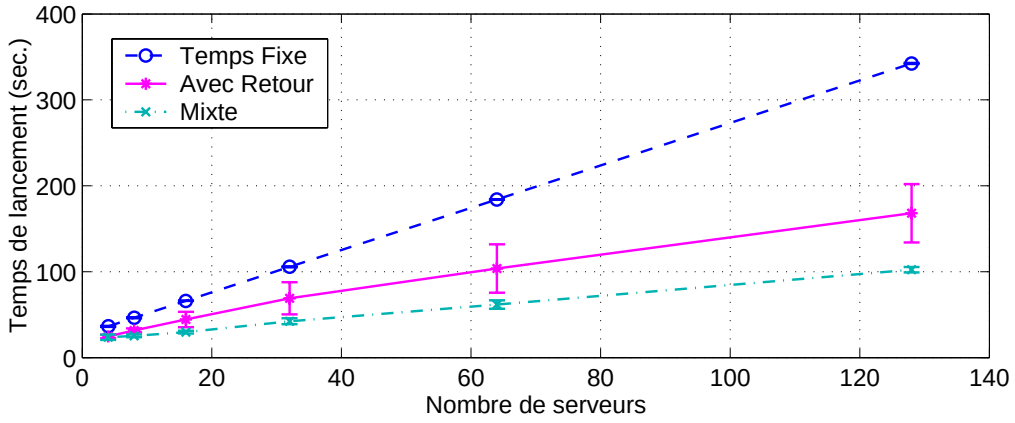


Figure 4: Temps de chargement de la plate-forme en fonction d'un nombre de Sed pour trois méthodes de chargement (temps fixe, avec retour et mixte).

5.2 Expérimentations sur le comportement de la plate-forme.

Le tableau 2 résume les autres données produites par nos expérimentations. On peut vérifier que le temps moyen d'une requête DIET est similaire avec les trois approches et augmente avec le nombre de serveurs. La seconde information nous donne le pourcentage de requêtes client qui n'ont pu aboutir, cette mesure correspond aux fautes dans la hiérarchie d'agents (une faute au niveau des serveurs n'apparaît pas au niveau des requêtes des clients vers les agents). Nous remarquons que ce type de fautes n'apparaît qu'avec la méthode mixte, et que le nombre de fautes n'est pas négligeable. Nous étudions ce problème, mais supposons que cela soit dû au chargement très rapide d'un trop grand nombre de serveurs. Ignorant ce problème jusqu'alors, aucun mécanisme pour gérer cette surcharge au niveau de la hiérarchie de DIET n'avait encore été mis en œuvre. La dernière métrique, correspond au pourcentage de serveurs inaccessibles par le client. Elle correspond au nombre de pannes (au sens large) au niveau des serveurs.

Pour conclure, l'approche mixte est la méthode la plus rapide mais elle manque de stabilité. Nous espérons dans des travaux futurs remédier à ce problème. En attendant, la méthode avec retour reste la méthode la plus intéressante, à la fois rapide et adaptée à la grille et à la diversité des réseaux.

	4 SeDs	8 SeDs	16 SeDs	32 SeDs	64 SeDs	128 SeDs
GoDIET	Latence moyenne des requêtes DIET (sec)					
Temps Fixe	0.50	0.63	0.69	1.04	1.59	2.73
Avec Retour	0.50	0.63	0.71	0.96	1.37	2.54
Mixte	0.33	0.57	0.63	1.09	1.43	2.08
	Pourcentage de requêtes non répondues					
Temps Fixe	0%	0%	0%	0%	0%	0%
Avec Retour	0%	0%	0%	0%	0%	0%
Mixte	0%	29.2%	0%	32.3%	33.3%	0%
	Pourcentage de serveurs inaccessibles par le client					
Temps Fixe	0%	0%	0%	0%	0%	0%
Avec Retour	0%	0%	2.1%	3.1%	0%	0%
Mixte	8.3%	29.2%	4.2%	33.3%	36.5%	3.4%

Table 2: Résultats de l'évaluation de la plate-forme logicielle.

6 Conclusion

Dans cet article nous avons présenté GoDIET, un outil de déploiement d'éléments distribués et hiérarchiques. Nous avons proposé trois méthodes (Temps Fixe, Avec retour, Mixte) permettant de prendre en considération les contraintes liées aux caractéristiques de nos éléments. Ces méthodes ont été évaluées expérimentalement sur le déploiement d'un intergiciel grille (DIET) et d'un service de gestion de traces (LogService). Les perspectives de GoDIET, se situent à deux niveaux, d'une part son intégration dans un outil de visualisation offrant ainsi à ce dernier un mode interactif permettant de gérer la plate-forme; d'autre part, nous envisageons de fournir une extension permettant d'adapter facilement GoDIET à d'autres types d'environnement.

References

[1] Gabriel Antoniu, Luc Bougé, Mathieu Jan, and Sébastien Monnet. Going large-scale in P2P experiments using the JXTA distributed framework. In *Euro-Par 2004: Parallel Processing*, number RR-5151 in Lect. Notes in Comp. Science, Pisa, Italy, August 2004. Springer-Verlag.

[2] E. Caron, S. Chaumette, S. Contassot-Vivier, F. Desprez, E. Fleury, C. Gomez, M. Goursat, E. Jeannot, D. Lazure, F. Lombard, J.-M. Nicod, L. Philippe, M. Quinson, P. Ramet, J. Roman, F. Rubi, S. Steer, F. Suter, and G. Utard. Scilab to scilab//, the ouragan project. *Parallel Computing*, 11(27):1497–1519, OCT 2001.

[3] Eddy Caron and Frédéric Desprez. Diet, tour d'horizon. In *Ecole thématique sur la Globalisation des Ressources Informatiques et des Données : Utilisation et Services. GridUSe 2004*, Metz. France, july 2004. Supélec.

[4] Henri Casanova, Graziano Obertelli, Francine Berman, and Rich Wolski. The AppLeS Parameter Sweep Template: User-level middleware for the Grid. In *Proceedings of Supercomputing*, November 2000.

[5] F. Desprez, M. Quinson, and F. Suter. Dynamic performance forecasting for network enabled servers in a metacomputing environment. In *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA 2001)*, 2001.

[6] LogService. http://graal.ens-lyon.fr/LogService.