# GoDIET: a deployment tool for distributed middleware on Grid'5000

Eddy Caron, Pushpinder Kaur Chouhan, Holly Dail

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

# *GoDIET: a deployment tool for distributed middleware on Grid'5000*

Eddy Caron — Pushpinder Kaur Chouhan — Holly Dail

**N° 5886**

April 2006

Thème NUM

ISSN 0249-6399 ISRN INRIA/RR--5886--FR+ENG

# GoDIET: a deployment tool for distributed middleware on Grid'5000

Eddy Caron , Pushpinder Kaur Chouhan , Holly Dail

Thème NUM — Systèmes numériques
Projet GRAAL

Rapport de recherche n° 5886 — April 2006 — 20 pages

**Abstract:** In this article we present GoDIET, a tool for the configuration, launch, and management of the Distributed Interactive Engineering Toolbox (DIET) on computational grids. DIET is an Application Service Provider (ASP) platform providing remote execution of computational problems on distributed resources. GoDIET automatically generates and stages all necessary configuration files, launches agents and servers in appropriate hierarchical order, reports feedback on the status of running components, and allows shutdown of all launched software.

GoDIET requires an XML file describing available compute and storage resources and the desired overlay of DIET agents and servers onto available resources. For homogeneous clusters, the XML file can be generated according to a deployment planning model, which has shown that an optimal DIET deployment on homogeneous clusters is a Complete Spanning $d$-ary (CSD) tree, where $d$ is the number of children directly attached to an agent, regardless of whether the children are servers or agents. We present experiments, that permit the evaluation of the performance of GoDIET for several launch and management approaches, that verify the correctness of the deployed platform, and that test the performance of CSD tree deployments optimized for uniform workloads under mixed workload scenarios.

**Key-words:** Deployment, ASP, Grid computing

Unité de recherche INRIA Rhône-Alpes
655, avenue de l'Europe, 38334 Montbonnot Saint Ismier (France)
Téléphone : +33 4 76 61 52 00 —Télécopie +33 4 76 61 52 52

# GoDIET : un outil de déploiement pour intergiciels distribués sur Grid'5000

**Résumé :** Dans cet article, nous exposons les travaux menés autour de la configuration, du lancement et de la gestion de Distributed Interactive Engineering Toolbox (DIET), un intergiciel de type Application Service Provider (ASP) 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'outil 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 utilisation 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. Enfin, nous avons expérimenté le déploiements d'arbre CSD, optimisés pour des charges de travail uniformes, dans un contexte de charges de travail non-uniformes.

**Mots-clés :** Déploiement, ASP, Calcul sur la grille

## 1 Introduction

The Distributed Interactive Engineering Toolbox (DIET) is an Application Service Provider (ASP) which is designed to simplify access to remote computational resources and software services for users. DIET is based on a hierarchy of distributed *agents* that collaborate to perform scheduling decisions, on computational *server daemons* that provide computational services, and on user-level *clients* that manage negotiation with the system for users.

In this paper we are interested in automated approaches for launching and managing hierarchies of DIET agents and servers across computational grids. The launch of DIET on computational grids presents a number of problems shared with many other grid solutions: how to quickly and reliably launch a large number of components, how to interface with a variety of resource environments (ranging from direct execution with `ssh` to launching via a batch system), how to regain control of launched processes to shut them down, and how to identify and react to failures in the launch process. The launch of DIET is further complicated by the fact that agents and servers must coordinate with the rest of the deployment; specifically, the chosen hierarchy is encapsulated in agent and server configuration files and the launcher must ensure that parent agents are fully functional before launching agents and servers that are further down in the hierarchy. We will use the word **deployment** to encapsulate all of the above general and DIET-specific launch and management issues; although deployment is an overloaded term, and thus not a desirable choice, there are scant alternatives for encapsulating these issues in one word.

Traditionally, users of DIET have either launched agents and servers by hand, or written scripts to manage the launch. These approaches place serious limitations on the diversity, scale, and number of experiments that can be feasibly performed. Since DIET itself is currently stable and provides good performance at scales up to 500 nodes and more, support for larger scale experiments is vital for further testing.

GoDIET is written in Java and users write an XML file describing their available compute and storage resources and the desired overlay of agents and servers onto those resources. GoDIET automatically generates and stages all necessary configuration files, launches agents and servers in appropriate hierarchical order, reports feedback on the status of running components, and manages shutdown of all launched software.

To facilitate users in generating an appropriate XML file, we are developing automated deployment planning models and approaches. We have demonstrated that a complete spanning $d$-ary tree (CSD tree) is an optimal deployment theoretically for DIET in large homogeneous cluster environments [6]. In the same paper we showed that we could correctly predict a good degree $d$ and that the selected CSD tree provided a good deployment in practice on large Grid'5000 clusters. Once the deployment planning module predicts a good $d$ for a particular environment and workload, we use another program to automatically generate an XML file for GoDIET which describes the exact mapping of the chosen CSD tree onto real resources. In this article we use this deployment model to test mixed jobs in DIET platforms.

The rest of this article is organized as follows. Section 2 presents related tools. In Section 3 we provide an overview of DIET and explain the DIET platform launch steps, Section 4 presents the deployment planning model for homogeneous clusters, and Section 5 describes in detail the deployment approach used by GoDIET. Then, in Section 6 we present experiments that show the performance of GoDIET. Finally, Section 7 concludes the paper and describes future work.

## 2 Related work

A variety of toolkits are freely available that can be used for launching programs on computational grids; we briefly discuss some of these toolkits below. We wanted GoDIET to satisfy some very specific DIET needs such as respecting a DIET hierarchy in the launch order, using LogService [4] feedback to manage the launch, and automatically generating configuration files that encapsulate each elements' role in the hierarchy. We did not find any currently existing toolkits that manage these tasks in a generalized way. Thus, rather than make very heavy modifications of an existing toolkit, we decided to purpose-build a deployment tool; however, we based on our approach on existing tools wherever possible.

JXTA Distributed Framework (JDF) [1] is designed to facilitate controlled testing of JXTA-based peer-to-peer systems under a variety of scenarios including emulated failures. JDF also uses an XML file for descriptions of the resources and jobs to be run, and deployment is based on a regular Java Virtual Machine, a Bourne shell, and `ssh` or `rsh`. JDF is focused on launching Java-based programs and relies on several Java features during the launch, thus it is not directly applicable to launching DIET. However, DIET has been adapted to use JuxMem [10], a JXTA-based distributed memory project. To enable this joint usage, GoDIET has been adapted to use existing JDF interfaces to coordinate the launch of JuxMem components in conjunction with the launch of DIET components. JDF also includes a notion of profiles that could be used to simplify the GoDIET XML by describing groups of agents or servers as a collection.

APST (AppLeS Parameter Sweep Template) [5] provides large-scale deployment of applications based on the parameter sweep model. Resources, jobs, and parameters are described in an XML file and APST manages the staging of files, scheduling of jobs, and retrieval of results. APST is a very general solution that can be used for a wide variety of purposes; APST is not however designed for middleware deployments structured with automatically-generated configuration files. Elagi [9] is a library that provides compiled programs with easy access to grid services; essentially, Elagi packages those parts of the APST code-base that can be easily re-used by other projects. We plan to rewrite those portions of GoDIET that use `ssh` and `scp` to instead use Elagi's generic process spawning and batch submission interfaces; this will provide GoDIET the ability to deploy DIET via `ssh`, `globusrun`, or a large number of batch schedulers including LSF, PBS, and Condor.

Automatic Deployment of Applications in a Grid Environment (ADAGE) [11] is a prototype middleware. It currently deploys only static applications on the resources of a computational grid. Deployed applications can be distributed applications (like CORBA component assembly), parallel applications (like MPICH-G2) or a combination of the two applications.

ADAGE is a promising approach. We plan to follow the development of this deployment approach to see where future integration and collaboration may be possible.

## 3 DIET overview

DIET [4] is a toolkit for building applications based on remote computational servers. The goal of DIET is to provide a sufficiently simple interface to mask the distributed infrastructure while providing users an access to greater computational power.

### 3.1 DIET architecture

DIET can use a hierarchy of agents to provide greater scalability for scheduling client requests on available servers. The collection of agents uses a broadcast / gather operation to find and select amongst available servers while taking into account locations of any existing data (which could be pre-staged due to previous executions), the load on available servers, and application-specific performance predictions.

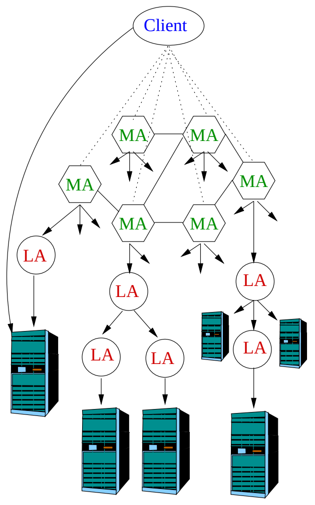


Figure 1: General view of DIET architecture.

An overview of the DIET architecture is shown in Figure 1. The *Master Agent* (MA) must be launched before all other agents and servers. The MA serves as the main *portal* to the DIET hierarchy; all agents and servers are thereafter attached to the MA via a tree at launch-time, though the type of tree is not constrained. All other agents are called *Local Agents* (LA) and can be included in the hierarchy for scalability or to provide a better mapping to the underlying network architecture. *Servers* (SeDs) are attached to each computational resource and either perform the actual computation directly or launch another binary to perform the computation (e.g. in the case of a server on a front-end node of a batch system). Servers also aid in the scheduling process by providing performance predictions to agents during the server selection process. While the top of the tree must be an MA, any number of LAs can be connected in an arbitrary tree and servers can be attached to the MA or to any of the LAs.

The *client* is the application interface by which users can submit problems to DIET. Many clients can connect to DIET at once and each can request the same or different types of services by specifying the problem to be solved. Clients can use either the synchronous interface where the client must wait for the response, or the asynchronous interface where the client can continue with other work and be notified when the problem has been solved.

Problem solution proceeds in two phases. In the *scheduling phase* the DIET client submits the problem to the MA. Agents maintain a simple list of subtrees containing a particular service; thus the MA broadcasts the request on all subtrees containing that service. Other agents in the hierarchy perform the same operation, forwarding the request on all eligible subtrees. When the request reaches the server-level, each server calls their local FAST [8] service to predict the execution time of the request on this server. These predictions are then passed back up the tree where each level in the hierarchy sorts the responses to minimize execution time and passes a subset of responses to the next higher level. Finally the MA returns one or several proposed servers to the client. In the *service phase* the client submits the problem directly to the chosen server. The server executes the job and sends the result back to the client.

This description of the agent/server architecture focuses on the common-case usage of DIET. An extension of this architecture is also available that uses JXTA peer-to-peer technology to allow the forwarding of client requests between MAs, and thus the sharing of work between otherwise independent DIET hierarchies. We are investigating the possible benefits of this peer-to-peer extension of DIET for scalability, fault-tolerance, and collaborations between administrative domains that desire greater independence.

## 3.2 DIET deployment

The DIET platform is constructed following the hierarchy of agents and SeDs; Figure 2 provides an overview of steps of the DIET startup process. The first element to be launched during deployment is the naming service; all other launched elements can easily find each other using only the hostname and port at which the naming service can be found and a string-based name for the element of interest. As a benefit of this approach, multiple DIET deployments can be launched on the same group of machines without conflict as long as the naming service for each deployment uses a different port and/or a different machine. After the naming service, the MA is launched; the MA is the root of the DIET hierarchy and thus does not register with any other elements. After the MA, all other DIET elements understand their place in the hierarchy from their configuration file which contains the name of the element's parent. Users of GoDIET specify the desired hierarchy in a more intuitive way via the naturally hierarchical XML input file to GoDIET (see Section 5).

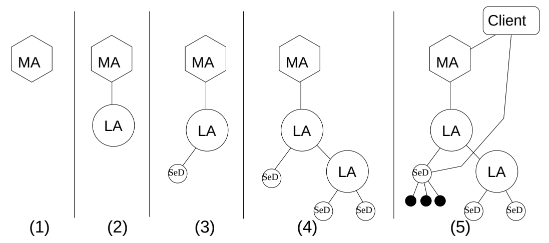


Figure 2: Launch of a DIET platform.

## 4 Deployment planning model

Deployment planning is the process of defining a good mapping of software components to available resources. For DIET, we consider that a *good deployment* is one that maximizes the steady-state throughput of the system, i.e. the number of requests that can be scheduled, launched, and completed by the servers in a given time unit. In DIET important issues for deployment planning include how many resources to allocate to agents (e.g. scheduling) versus to servers (e.g. computation) and what hierarchical arrangement of agents and servers to use. In [3] we presented a heuristic approach to improving an existing hierarchical deployment on heterogeneous resources. The approach is iterative; in each iteration, mathematical models are used to analyze the existing deployment, identify the primary bottleneck, and remove the bottleneck by adding resources in the appropriate area of the system. In [6] we presented an approach for determining an optimal hierarchical middleware deployment for a homogeneous resource platform of a given size. We showed that an optimal deployment for hierarchical middleware systems on clusters is provided by a Complete Spanning $d$-ary (CSD) tree. The approach automatically determines an appropriate degree $d$ and the number of nodes that should be used so as to maximize the steady-state throughput.

We have validated this theoretical work for homogeneous requests on homogeneous clusters in [6]. In this article we test the performance of CSD trees optimized for homogeneous workloads under mixed workload conditions.

In order to apply the deployment planning model [6] to DIET, we have to define models for the scheduling phase and the service phase of DIET. Here we consider only the single port serial model wherein a computing resource has no capability for parallelism: it can either send a message, receive a message, or compute. Messages must be sent serially and received serially. The throughput of the scheduling phase in requests per second is given by the minimum of the throughput provided by the servers for prediction and by the agents for scheduling the request. As only servers take part in the service phase, the throughput of the service phase in requests per second is given by the throughput provided by the servers for service provision, which can be affected by the servers' participation in the prediction phase.

## 5 GoDIET

GoDIET is designed to automate the deployment of DIET platforms and associated services for diverse grid environments. Key goals of GoDIET included portability, the ability to integrate GoDIET in a graphically-based user tool for DIET management, and the ability to communicate in CORBA with LogService [4]; we have chosen Java for the GoDIET implementation as it satisfies all of these requirements and provides for rapid prototyping. Figure 3 shows how GoDIET works in conjunction with services for logging and visualization to provide convenient administration and monitoring for a running DIET deployment.

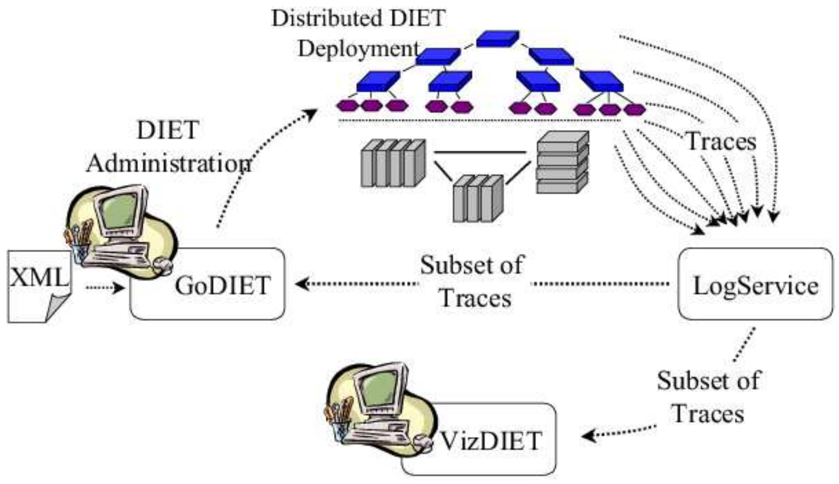


Figure 3: Interaction of GoDIET, LogService, and VizDIET to assist users in controlling and understanding DIET platforms.

GoDIET automatically generates configuration files for each DIET component taking into account user configuration preferences and the hierarchy defined by the user, launches complimentary services (such as a name service and logging services), provides an ordered launch of components based on dependencies defined by the hierarchy, and provides remote cleanup of launched processes when the deployed platform is to be destroyed.

An important aspect of such a platform administration tool is the ability to identify and react to errors in the deployment process. When GoDIET identifies an error in the launch of a particular element, it analyzes the elements of the hierarchy that have not yet been launched and only launches those elements that do not depend on the failed element. For example, if the naming service launch fails, GoDIET will not launch any other elements because all other elements depend on this service. This reactivity saves the user time because time is not wasted trying to launch elements that will fail anyway.

GoDIET can identify errors in the launch process in two ways. First, if errors are reported on standard error during the launch of an element, GoDIET reports these errors to the user and marks the launch of the element as *confused* because GoDIET can not be sure of the status of the element; unlaunched elements are marked with a launch status of *none* and correctly launched elements have a status of *running*. Second, if the user has requested that LogService be launched, GoDIET registers with LogService for traces concerning all running elements. Thus, after the launch of each element GoDIET waits for feedback from LogService concerning the health of the launched element. If LogService does not report that an element has launched successfully, GoDIET marks the log state of that element as *confused*. The verification of launch state is a useful tool for monitoring the health of the deployment because it is always available regardless of whether the user chooses to deploy LogService or not. However, the availability of LogService provides a much stronger verification on launch state because many more types of errors can be caught using this log feedback.

As input GoDIET requires an XML file, in which users describe their available compute and storage resources and the desired overlay of agents and servers onto those resources. In short, the GoDIET XML file contains the description of DIET agents and servers and their hierarchy, the description of desired complementary services like LogService, the physical machines to be used, the disk space available on these machines, and the configuration of paths for the location of needed binaries and dynamically loadable libraries. The file format provides a strict separation of the resource description and the deployment configuration description; the resource description portion must be written once for each new grid environment, but can then be re-used for a variety of deployment configurations.

Detailed and well commented XML examples are presented in the GoDIET documentation. We use a Document Type Definition file (DTD) to provide automated enforcement of allowed XML file semantics; an input XML file is verified against the GoDIET DTD using a validating parser. An example XML file is given in the Appendix A.

From the Appendix A it can be seen that `diet_configuration` markups surround all of the other sections. `diet_configuration` can optionally contain a "goDiet" section to allow configuration of GoDIET behavior. `diet_configuration` must contain three subsections: the `resources` section, the `diet_services` section and the `diet_hierarchy` section.

The `resources` section defines what machines to use for computation and storage, how to access those resources, and where to find binaries and libraries on each machine. It must include at least one "scratch" section, one "storage" section, and one "compute" or one "cluster" section. The `diet_services` section must contain one "omni_names" section and can optionally include one “log_central” and one “log_tool” section. The `diet_hierarchy` section defines the deployment hierarchy and must include at least one “master_agent” section and one “SeD” section.

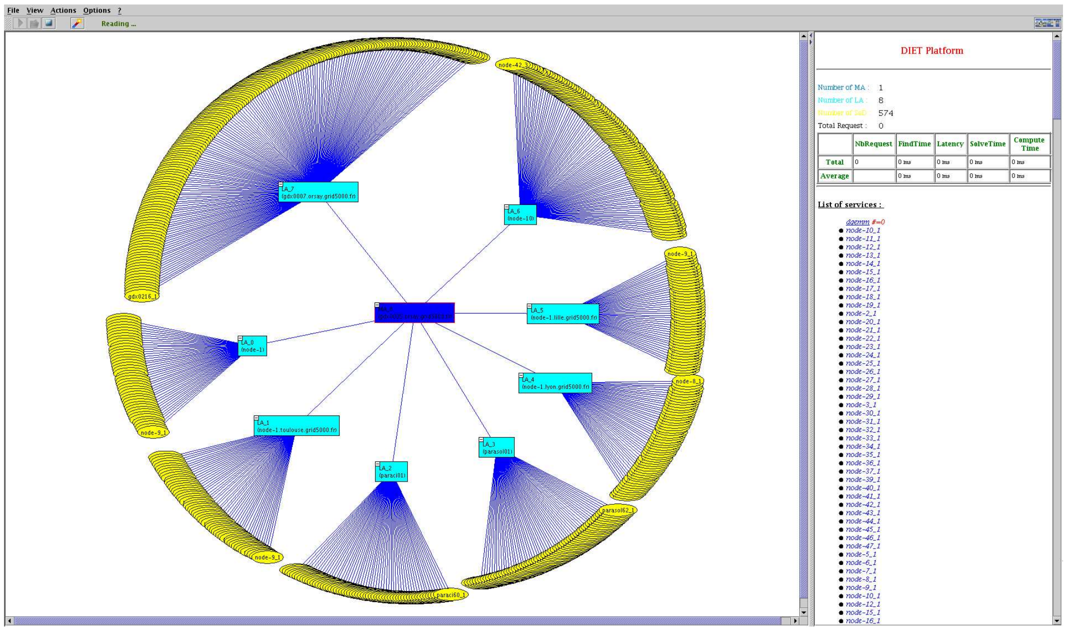
The basic user interface is a non-graphical console mode and can be used on any machine where Java is available and where the machine has `ssh` access to the target resources used in the deployment. An alternative interface is a graphical console that can be loaded by VizDIET [2] to provide an integrated management and visualization tool. Both the graphical and non-graphical console modes can report a variety of information on the deployment including the run status and, if running, the PID of each component, as well as whether log feedback has been obtained for each component.

We use `scp` and `ssh` to provide secure file transfer and task execution. `ssh` is a tool for remote machine access that has become almost universally available on grid resources in recent years. With a carefully configured `ssh` command, GoDIET can configure environment variables, specify the binary to launch with appropriate command line parameters, and specify different files for the stdout and stderr of the launched process. Additionally, for a successful launch GoDIET can retrieve the PID of the launched process; this PID can then be used later for shutting down the DIET deployment. In the case of a failure to launch the process, GoDIET can retrieve these messages and provide them to the user. To illustrate the approach used, an example of the type of command used by GoDIET follows.

```
/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_exp01/omniORB4.cfg; ✂
   cd /home/user/godiet_s/run_exp01; ✂
   nohup dietAgent ./MA_0.cfg < /dev/null > MA_0.out 2> MA_0.err &" ; ✂
   /bin/echo '/bin/echo ${!}' )  ✂
   | /usr/bin/ssh -q user@grid5000.ens-lyon.fr /bin/sh -
```

It is important that each `ssh` connection can be closed once the launch is complete while leaving the remote process running. If this can not be achieved, the machine on which GoDIET is running may eventually run out of resources (typically sockets) and refuse to open additional connections. In order to enable a scalable launch process, the above command ensures that the `ssh` connection can be closed after the process is launched. Specifically, in order for this connection to be closeable: (1) the UNIX command `nohup` is necessary to ensure that when the connection is closed the launched process is not killed as well, (2) the process must be put in the background after launch, and (3) the redirection of all inputs and outputs for the process is required.

DIET provides the features and flexibility to allow a wide variety of deployment configurations, even in difficult network and firewall environments. For example, for platforms without DNS-based name resolution or for machines with both private and public network interfaces, elements can be manually assigned an *endpoint* hostname or IP in their configuration files; when the element registers with the naming service, it specifically requests this endpoint be given as the contact address during name lookups. Similarly, an *endpoint* port can be defined to provide for situations with limited open ports in firewalls. These specialized options are provided to DIET elements at launch time via their configuration files; GoDIET supports these configuration options via more user-intuitive options in the input XML file and then automatically incorporates the appropriate options while generating each element's configuration file. For large deployments, it is key to have a tool like GoDIET to make practical use of these features.



DIET Platform

Number of MA : 1
Number of LA : 8
Number of SeD : 574
Total Request : 0

| | NbRequest | FindTime | Latency | SolveTime | Compute Time |
|---|---|---|---|---|---|
| **Total** | 0 | 0 ms | 0 ms | 0 ms | 0 ms |
| **Average** | | 0 ms | 0 ms | 0 ms | 0 ms |

**List of services :**

*dgemm* *#=0*
- *node-10_1*
- *node-11_1*
- *node-12_1*
- *node-13_1*
- *node-14_1*
- *node-15_1*
- *node-16_1*
- *node-17_1*
- *node-18_1*
- *node-19_1*
- *node-2_1*
- *node-20_1*
- *node-21_1*
- *node-22_1*
- *node-23_1*
- *node-24_1*
- *node-25_1*
- *node-26_1*
- *node-27_1*
- *node-28_1*
- *node-29_1*
- *node-3_1*
- *node-30_1*
- *node-31_1*
- *node-32_1*
- *node-33_1*
- *node-34_1*
- *node-35_1*
- *node-36_1*
- *node-37_1*
- *node-39_1*
- *node-40_1*
- *node-41_1*
- *node-42_1*
- *node-43_1*
- *node-44_1*
- *node-45_1*
- *node-46_1*
- *node-47_1*
- *node-5_1*
- *node-6_1*
- *node-7_1*
- *node-8_1*
- *node-9_1*
- *node-10_1*
- *node-12_1*
- *node-15_1*
- *node-16_1*

Figure 4: Grid'5000 DIET Deployment with 1 MA, 8 LA, 574 SeD

## 6 Experiments

We present experiments designed to evaluate the performance and efficacy of GoDIET for the deployment of DIET and associated services at a large scale, the accuracy of GoDIET in identifying errors in the deployment, and the performance of deployments optimized for homogeneous workloads under mixed workload scenarios.

We did experiments on different sites of Grid'5000, a set of distributed computational resources in France. Table 1 provides details of the Grid'5000 sites that we used for our experiments. An abstract example of an XML file used for experiments is shown in Appendix

A. All tests were performed using the DGEMM application, a simple matrix multiplication provided as part of the Basic Linear Algebra Subprograms (BLAS) package [7].

## 6.1 Evaluation of launch performance

For this experiment we want to test the time required by GoDIET to launch DIET deployments of different sizes. A key activity for GoDIET during launch is to determine when an element has finished launching and registering itself with the naming service; only once these processes are done can GoDIET launch dependent elements. There are two approaches used to define timing of dependent element launch.

**Fixed wait:** This is the simplest approach, and involves simply sleeping for a fixed period before launching dependent components. Our experiments are performed with a wait of 3 seconds after omniNames and LogCentral, 2 seconds after each agent, and 1 second after each server.

**Feedback:** This approach uses real-time feedback from LogService to guide the launch process. GoDIET waits for verification that a component has registered with the logging service before launching other components.

For these experiments, the type of DIET hierarchy is fixed and we vary the number of sites (and therefore servers) to be deployed. We obtained access to a subset of the machines at each cluster listed in Table 1: 30 nodes at Lyon, 40 nodes at Bordeaux, 40 nodes at Lille, 50 nodes on the Rennes Paraci cluster, 50 nodes at Toulouse, 90 nodes at Sophia, and 140 nodes at Orsay; thus we had 440 nodes in total. We defined our hierarchy to be composed of an MA on the cluster Lyon and an LA on each of the other sites in the deployment; all remaining nodes were allocated as SeDs attached to the LA at their site. To test the performance of GoDIET for different deployment sizes, we first test a deployment using only the smallest site (1 MA, 1 LA, and 28 SeDs total), then a deployment using the three smallest sites (1 MA, 3 LAs, and 102 SeDs total), and so on until we have tested a deployment using all 7 of the sites (1 MA, 7 LAs, and 426 SeDs total). We test the time required for GoDIET

| Site Cluster | Nodes | Memory | Processor Type |
|---|---|---|---|
| Bordeaux | 48 | 2 GB | dual AMD Opteron 248 2.2GHz |
| Lille | 53 | 4 GB | dual AMD Opteron 248 2.2GHz |
| Lyon | 56 | 2 GB | dual AMD Opteron 2.0 |
| Orsay | 216 | 2 GB | dual AMD Opteron 246 2.0GHz |
| Rennes Paraci | 64 | 2 GB | dual Intel Xeon 2.4 GHz |
| Sophia | 138 | 2 GB | dual AMD Opteron 246 2.0GHz |
| Toulouse | 57 | 2 GB | dual AMD Opteron 248 2.2GHz |

Table 1: Description of the Grid'5000 clusters used in our experiments.

to launch each of these deployments using the *fixed wait* approach and using the *feedback* approach. Figure 5 presents the time required to launch these different platforms.

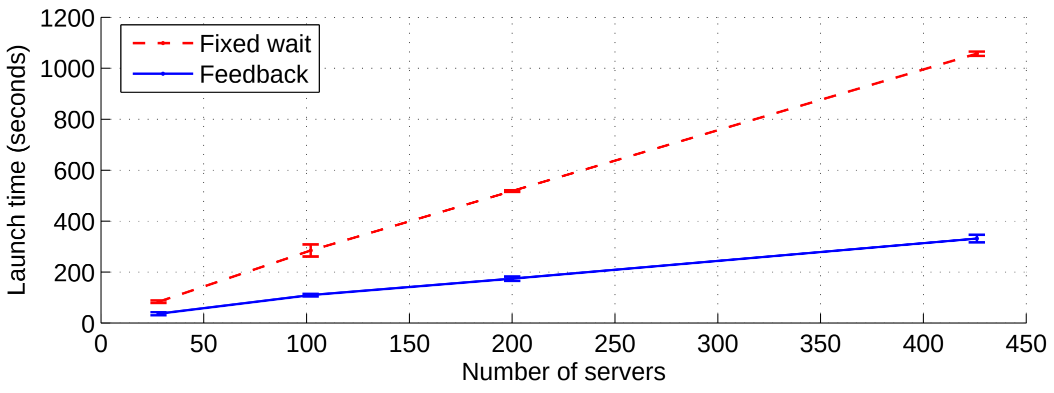


Figure 5: The time for platform launch as a function of the number of servers desired.

The time for launch is strongly dependent on the number of servers included in the deployment. The cost of the *fixed wait* approach is clearly high relative to the cost of the *feedback* approach in this environment. Grid'5000 benefits from top-level machines and networks and relatively little competition for resources among users; in this environment our selection of fixed wait times of several seconds between elements is likely overgenerous and the *feedback* approach receives notifications very quickly. However, in more difficult network environments these times may be too short. The *feedback* approach is clearly preferable because it automatically controls dependent element launch based on feedback from previous launches and so does not require user intervention to guess the correct amount of wait time.

## 6.2 Launch problem identification

Once a platform has been launched by GoDIET, we want to verify that the platform was indeed correctly launched. We are also interested in how accurately GoDIET identifies problems in the platform based on launch-time errors and/or LogService feedback. We the following platform verifications on all deployments from Section 6.1.

For these experiments we take advantage of one of the DIET scheduling modes that provides round-robin allocation of tasks to SeDs. To test the performance of a deployment with $S$ servers, we run $S$ clients against that deployment. If we have $S$ valid scheduler responses from the agent hierarchy, then we consider that the agent hierarchy is running well. We then verify that each of the $S$ requests was assigned a unique server (as required by our choice of round-robin). If so, and if the servicing phase of all $S$ requests completed successfully, we consider all $S$ servers to be functioning correctly. If less than $S$ unique servers were used to service the $S$ requests, we know some servers or their parent agents were not functioning correctly.

Note that DIET provides best-effort round-robin scheduling and under cases of high rates of arrival of competing client requests, sometimes DIET is not able to provide strict round-robin behavior. Thus, to ensure strict round-robin behavior our testing script only launches a single client at a time; under these conditions, round-robin behavior is strictly enforced by DIET.

For each deployment we also identified how many errors GoDIET reported to the user, for comparison against the number of errors seen via the above verification test. In the *fixed wait* approach a server is marked as failed if its launch seemed to have failed, while in the *feedback* approach a server is marked as failed if either its launch failed or log feedback was received for it.

Table 2 provides a summary of our analysis of problematic SeDs in the deployments from the previous section. The majority of deployments had no errors at all: all agents and SeDs were functioning correctly; this result is promising in terms of the stability and usability of Grid'5000 since such large resource sets typically have some number of unexpected run-time problems. However, when there are problematic SeDs, GoDIET does not effectively estimate the scale of the problem. In fact, when there are problems they are often large with as many as 129 problematic SeDs. In all cases, these large numbers of problems correspond to the size of one of the individual sites in our deployment, so we believe that these errors are site-level errors corresponding to a failed local agent at that site or a transitory problem with the site itself (such as a temporary network partition). In any case, we plan to improve GoDIET to better identify these problems so that DIET administrators and users can be better informed about the platform status.

## 6.3 Large scale experiment

From the results of different GoDIET launch approaches, we have selected the feedback approach to deploy DIET on a hierarchy of 585 nodes. This experiment was done on 7

| Approach | Sites | Rep. 1 | Rep. 2 | Rep. 3 |
|---|---|---|---|---|
| Fixed wait | 1 | 0/0 | 0/0 | 0/0 |
| | 3 | 0/0 | 0/0 | 1/40 |
| | 5 | 1/50 | 0/49 | 0/39 |
| | 7 | 2/91 | 1/129 | - |
| Feedback | 1 | 0/0 | 0/0 | 0/0 |
| | 3 | 1/1 | 0/0 | 0/0 |
| | 5 | 0/0 | 0/0 | 0/0 |
| | 7 | 2/30 | 1/29 | - |

Table 2: Problem SeDs as identified by GoDIET / by clients.

| Cluster | SeDs | Launch time (secs) |
|---|---|---|
| Bordeaux | 44 | 70 |
| Lille | 44 | 118 |
| Lyon | 49 | 52 |
| Orsay | 176 | 88 |
| Paraci | 59 | 82 |
| Parasol | 59 | 81 |
| Sophia | 89 | 68 |
| Toulouse | 54 | 33 |

Table 3: Time taken to launch 574 SeDs is 592 secs.

sites/8 clusters (Bordeaux, Lille, Lyon, Orsay, Rennes, Sophia, Toulouse) of Grid'5000. OmniNames and log_central were launched at Orsay. The hierarchy is composed of an MA on the cluster Orsay and an LA on each of the 8 clusters. All other reserved nodes were added as servers to the LA on that cluster. GoDIET was located on a machine in the Lyon cluster. The time taken to launch this deployment is *7minutes 25secs*. The time taken to launch SeDs on each cluster is shown in Table 3.

## 6.4 Platform selection by deployment planning model

In this section we test whether it is better to deploy one DIET hierarchy on all available nodes and submit all (heterogenous) requests to the hierarchy, or whether it is better to divide the available nodes in different partitions with one partition per problem size. For example, if we have 100 nodes and four different problems, should we deploy one CSD tree using all 100 nodes to solve the four types of problems, or should we deploy four CSD trees of 25 nodes each and submit only uniform problems to each CSD tree.

For this experiment we use a total of 75 nodes at the Orsay site and DGEMM problem sizes 10, 100, and 1000. First, we divided the available nodes in three sets of 25 nodes each. Then we used our optimal deployment planning algorithm to predict the best value of $d$ to construct a CSD tree with 25 nodes for each problem size. Next, we used the planning algorithm to predict the best value of $d$ for a CSD tree using 75 nodes and each problem size.

For a deployment of size 25 nodes, for sizes 10 and 100 our algorithm predicts a best degree of 2, while for size 1000 the best degree is predicted to be 24. For a deployment of size 75 nodes, degree 2 is again predicted to be optimal for sizes 10 and 100 while degree 74 is predicted to be optimal for size 1000. Next, we tested the makespan of each set of tasks on the three separate deployments, sending only the appropriate problem size to each deployment. Then we tested the makespan of the combined set of tasks including all three problem sizes on the 75 node deployments. The results are shown in Figure 6.

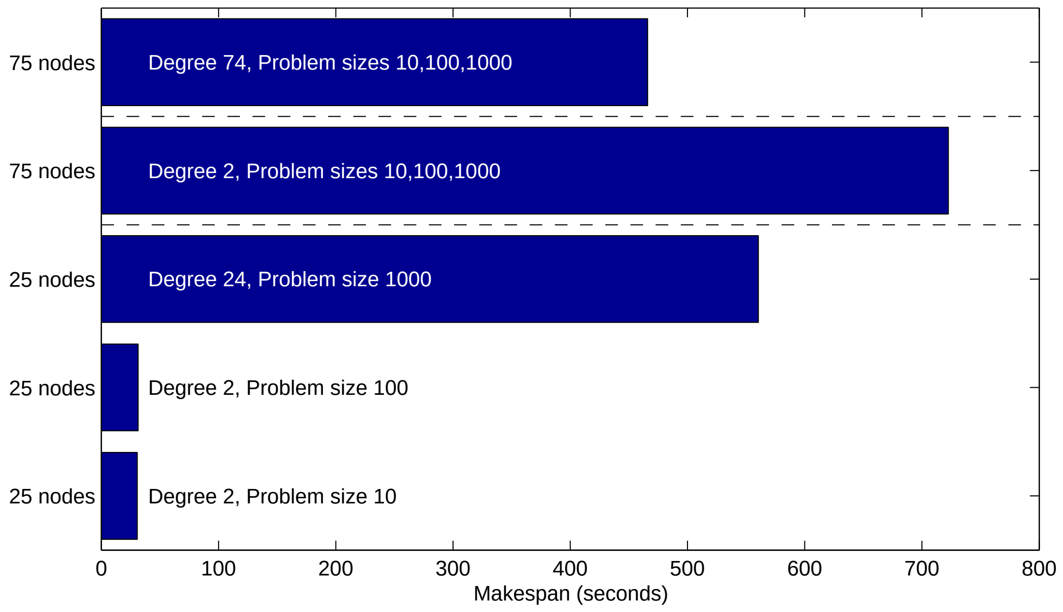


Figure 6: Makespan for a group of tasks partitioned to three deployments or sent to a single joint deployment.

Figure 6 shows that if we deploy three small CSD tree in parallel it take 560.36 seconds to execute 3000 requests but if we deploy one big CSD tree but with degree 74, we can save time, as it took only 466 seconds to execute 3000 requests.

## 7 Conclusion

In this article we have presented GoDIET, a tool for the hierarchical deployment of distributed DIET agents and servers. We described DIET and discussed in detail the launch process for DIET, explained some important features of GoDIET, and briefly described our deployment planning algorithms for homogeneous clusters. We then used experiments to show that GoDIET can effectively manage deployments of hundreds of nodes and that using feedback to guide the launch is an effective strategy. We also tested the launched deployments to verify that they were correctly launched and responsive to client requests. We identified some problems in failed SeDs in large-scale deployments. Finally we tested mixed workloads on CSD tree deployments optimized for homogeneous tasks; we found that, in our test, partitioning machines into deployments that are specific to a particular kind of workload may not be an effective approach.

There are several areas we plan to pursue in future work. First, we plan to build a visual tool that will allow users to build a graphical model of their desired deployment and export this model to GoDIET for launch. We also envision enabling interactive deployment and reconfiguration whereby users can dynamically reconfigure portions of the DIET deployment. Second, we plan to integrate this tool with the output obtained from the deployment planning model, which is based on resource and application characteristics. We also plan to extend and adapt our deployment planning approaches to usage in real-world deployment scenarios.

## References

[1] G. Antoniu, L. Bougé, and M. Jan. Juxmem: An adaptive supportive platform for data sharing on the grid. In *IEEE/ACM Workshop on Adaptive Grid Middleware, held in conjunction with 12th Intl. Conf. on Parallel Architectures and Compilation Techniques (PACT 2003)*, New Orleans, September 2003.

[2] R. Bolze, E. Caron, and F. Desprez. A monitoring and visualization tool and its application for a network enabled server platform. In LNCS, editor, *Parallel and Distributed Computing Workshop of ICCSA 2006*, Glasgow, UK., 8-11 May 2006.

[3] E. Caron, P. K. Chouhan, and A. Legrand. Automatic Deployment for Hierarchical Network Enabled Server. In *The 13th Heterogeneous Computing Workshop (HCW 2004)*, Santa Fe. New Mexico, April 2004.

[4] E. Caron and F. Desprez. DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid. *International Journal of High Performance Computing Applications*, 2006. To appear.

[5] H. Casanova, G. Obertelli, F. Berman, and R. Wolski. The AppLeS Parameter Sweep Template: User-level middleware for the Grid. In *Proceedings of Supercomputing*, November 2000.

[6] P.K. Chouhan, H. Dail, E. Caron, and F. Vivien. Automatic middleware deployment planning on clusters. *International Journal of High Performance Computing Applications*, 2007. To appear.

[7] A. Chtchelkanova, J. Gunnels, G. Morrow, J. Overfelt, and R. Van de Geijn. Parallel implementation of BLAS: General techniques for level 3 BLAS. Technical Report CS-TR-95-40, University of Texas, Austin, Oct. 1995.

[8] 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.

[9] Elagi project description. http://grail.sdsc.edu/projects/elagi/.

[10] JuxMem project description. http://www.jxta.org/universities/-juxmem.html.

[11] S. Lacour, C. Pérez, and T. Priol. A software architecture for automatic deployment of corba components using grid technologies. In *Proceedings of the 1st Francophone Conference On Software Deployment and (Re)Configuration (DECOR 2004)*, Grenoble, France, October 2004.

## A Example GoDIET XML file

```
<?xml version="1.0" standalone="no"?>
<!DOCTYPE diet_configuration SYSTEM "devel/GoDIET-2.0.0/GoDIET.dtd"
  >
<diet_configuration>
  <goDiet debug="1" saveStdOut="no" saveStdErr="no" useUniqueDirs="
      no"/>
  <resources>
    <scratch dir="/homePath/user/scratch_godiet"/>
    <storage label="g5kBordeauxDisk">
      <scratch dir="/homePath/user/scratch_runtime"/>
      <scp server="frontale.bordeaux.grid5000.fr" login="pkchouhan"
          />
    </storage>
    .
    .
    <storage label="g5kOrsayDisk">
      <scratch dir="/homePath/user/scratch_runtime"/>
      <scp server="frontale.orsay.grid5000.fr" login="pkchouhan"/>
    </storage>
    <cluster label="g5kBordo" disk="g5kBordeauxDisk" login="
        pkchouhan">
      <env path="/homePath/user/demo/bin" LD_LIBRARY_PATH="/
          homePath/user/demo/lib"/>
      <node label="node-1.bordeaux.grid5000.fr">
        <ssh server="node-1.bordeaux.grid5000.fr"/>
      </node>
                .
                .
      <node label="node-47.bordeaux.grid5000.fr">
          <ssh server="node-47.bordeaux.grid5000.fr"/>
      </node>
    </cluster>
          .
          .
    <cluster label="g5kOrsay" ..>
           .
           .
    </cluster>
  </resources>
  <diet_services>
    <omni_names contact="gdx0005.orsay.grid5000.fr" port="2809">
        <config server="gdx0005.orsay.grid5000.fr" remote_binary="
            omniNames"/>
    </omni_names>
    <log_central>
        <config server="gdx0007.orsay.grid5000.fr" remote_binary="
            LogCentral"/>
    </log_central>
    <log_tool>
        <config server="gdx0007.orsay.grid5000.fr" remote_binary="
            DIETLogTool"/>
    </log_tool>
  </diet_services>
  <diet_hierarchy>
    <master_agent label="MA1">
      <config server="node-5.toulouse.grid5000.fr" remote_binary="
           dietAgent"/>
      <local_agent label="LA1">
        <config server="node-75.sophia.grid5000.fr" remote_binary="
             dietAgent"/>
        <SeD label="SeD1">
            <config server="node-65.sophia.grid5000.fr"
                 remote_binary="BLASserver"/>
        </SeD>
            .
            .
        <SeD label="SeD89">
            <config server="node-34.sophia.grid5000.fr"
                 remote_binary="BLASserver"/>
        </SeD>
      </local_agent>
         .
         .
      <local_agent label="LA8">
        <config server="node-30.lyon.grid5000.fr"
                .
                .
      </local_agent>
    </master_agent>
  </diet_hierarchy>
</diet_configuration>
```

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399