Laboratoire de l'Informatique du Parallélisme

École Normale Supérieure de Lyon
Unité Mixte de Recherche CNRS-INRIA-ENS LYON-UCBL n^{o} 5668

Use of A Network Enabled Server System for a Sparse Linear Algebra Grid Application.

Eddy Caron,
Frédéric Desprez,
Christophe Hamerling,

Jean-Yves L'Excellent,
Marc Pantel,
Chiara Puglisi-Amestoy

June 2005

Research Report N^{o} 2005-30

Use of A Network Enabled Server System for a Sparse Linear Algebra Grid Application.

Eddy Caron, Frédéric Desprez, Christophe Hamerling,
Jean-Yves L'Excellent, Marc Pantel, Chiara Puglisi-Amestoy

June 2005

Abstract

Solving systems of linear equations is one of the key operations in linear algebra. Many different algorithms are available in that purpose. These algorithms require a very accurate tuning to minimise runtime and memory consumption. The TLSE project provides, on one hand, a scenario-driven expert site to help users choose the right algorithm according to their problem and tune accurately this algorithm, and, on the other hand, a test-bed for experts in order to compare algorithms and define scenarios for the expert site. Both features require to run the available solvers a large number of times with many different values for the control parameters (and maybe with many different architectures). Currently, only the grid can provide enough computing power for this kind of application. The DIET middleware is the GRID backbone for TLSE. It manages the solver services and their scheduling in a scalable way.

Keywords: Grid Computing, Sparse Linear System Solvers, Expert Site Framework.

Résumé

La résolution de systèmes linéaires creux est une opération clé en algèbre linéaire. Beaucoup d'algorithmes sont utilisés pour cela, qui dépendent de nombreux paramètres, afin d'offrir une robustesse, une performance et une consommation mémoire optimales. Le projet GRID-TLSE fournit d'une part, un site d'expertise basé sur l'utilisation de scénarios pour aider les utilisateurs à choisir l'algorithme qui convient le mieux à leur problème ainsi que les paramètres associés; et d'autre part, un environnement pour les experts du domaine leur permettant de comparer efficacement des algorithgmes et de définir dynamiquement de nouveaux scénarios d'utilisation. Ces fonctionalités nécessitent de pouvoir exécuter les logiciels de résolution disponibles un grand nombre de fois, avec beaucoup de valeurs différentes des paramètres de controle (et éventuellement sur plusieurs architectures de machines). Actuellement, seule la grille peut fournir la puissance de calcul pour ce type d'applications. L'intergiciel DIET est utilisé pour gérer la grille, les différents services, et leur ordonnancement efficace.

Mots-clés: Calcul sur Grille, Solveurs de Systèmes Linaires Creux, Site d'Expertise

1 Introduction

Large problems coming from numerical simulation or life science can now be solved through the Internet using grid middleware. Several approaches co-exist to port applications on grid platforms like object-oriented languages, classical message-passing, batch processing, web portals, etc.

Among existing middleware approaches, one simple, performant, and flexible approach consists in using servers available in different administrative domains through the classical client-server or Remote Procedure Call (RPC) paradigm. Network Enabled Servers, such as NetSolve[1] or Ninf[2], implement this model also called GridRPC [27]. Clients submit computation requests to a scheduler whose goal is to find a server available on the grid. Scheduling is frequently applied to balance the work among the servers and a list of available servers is sent back to the client; the client is then able to send the data and the request to one of the suggested servers to solve their problem. Thanks to the growth of network bandwidth and the reduction of network latency, small computation requests can now be sent to servers available on the grid. To make effective use of today's scalable resource platforms, it is important to ensure scalability in the middleware layers as well.

The Distributed Interactive Engineering Toolbox (DIET[3]) project [10] is focused on the development of a scalable middleware by distributing the scheduling problem across multiple agents. DIET consists of a set of elements that can be used together to build applications using the GridRPC paradigm. This middleware is able to find an appropriate server according to the information given in the client's request (problem to be solved, size of the data involved), the performance of the target platform (server load, available memory, communication performance) and the local availability of data stored during previous computations. The scheduler is distributed using several collaborating hierarchies connected either statically or dynamically (in a peer-to-peer fashion). Data management is provided to allow persistent data to stay within the system for future re-use. This feature avoids unnecessary communication when dependences exist between different requests.

Many applications are based on linear algebra kernels that are sometimes hard to install and tune for specific usages. This is usually the case for sparse linear algebra codes, with many different solutions depending on the functionality required (system of linear equations, linear least squares, etc.) and the kind of matrix used. The use of such libraries can be a problem and an external assistance is often needed: the expert work consists in analysing the properties of the problem, define which types of algorithms and software solutions might be applied, experiment them with various algorithmic parameters relevant to both the chosen package and the problem characteristics, in order to provide an answer to the end user. The goal of the Grid-TLSE (Test for Large Systems of Equations) project is to automatize this expertise using scenarios answering to common users needs, and thus help users in choosing the right algorithm depending on their problem and in tuning this algorithm by providing adequate input parameters. Since this requires to run the various solvers a significant number of times with many different values for the control parameters, grid computing is used, providing enough computing power for this kind of application. Note that the goal here is to run existing solvers on various architectures, and there is no plan to parallelise a given solver over the grid.

This paper is organised as follows. In a first section, we present Grid-TLSE, our motivating application, and its needs for grid computing. Then, in Section 3, we present the architecture of DIET, the middleware used to solve a large number of problems on dedicated servers. In Section 4, we discuss the overall architecture of the Grid-TLSE project and its design choices. Section 5 presents some related work. Finally, we conclude and discuss our future work.

[1]http://www.cs.utk.edu/netsolve
[2]http://ninf.etl.go.jp
[3]http://graal.ens-lyon.fr/DIET

2 Motivating Application

The Grid-TLSE project is a three-year project started in December 2002 and funded by the French Ministry of Research ACI GRID Program [1, 14]. The academic partners involved in this project are: CERFACS (Toulouse), IRIT (Toulouse), LaBRI (Bordeaux), and LIP-ENS (Lyon). These teams have been working together over many years in the field of sparse matrix computations and have a strong collaboration with the international scientific community that has given rise to the production of several software packages available to external users. End users are usually specialists of physical modelling and know only a strict minimum about numerical computation and even less about parallel and distributed applications. They usually encounter problems in order to choose the right tool according to, on one hand, their own constraints (physical application modelling, structural and numerical properties of the problem, architecture and performance of available computers and systems), and on the other hand, the selected tool constraints (required libraries, languages and operating system, algorithm tuning (numerical, parallelism, distribution)).

Users usually require assistance from tool developers who spend a huge amount of time providing it. Developers have designed strategies to offer help which involve many solver runs with many different parameter values on many different computer architectures. Grid-TLSE aims at automatizing these strategies (also referred to as expertise scenarios), to answer to specific users objectives. Developers also need to: compare the various algorithms on a given problem in order to provide insights on the most adapted one; compare the same algorithm on the same problem with various values for the problem parameters; combine various algorithms; and finally manage the results from all these runs. Grid-TLSE provide a framework in order to ease all these tasks.

Strong of their experience, these research teams decided to build a web site giving an easy access to tools and allowing scenario-driven comparative synthetic analysis of these packages. The site will be validated by industrial partners (CEA, CNES, EADS, EDF, and IFP).

2.1 Services Provided by Grid-TLSE

Three kinds of users may be interested in Grid-TLSE. First, **end users**, with either basic, medium or advanced knowledge in numerical computations and parallel/distributed programming. They mainly want to choose the best solver for their problem according to specific metrics (memory usage, robustness, accuracy, execution time) and find out the control parameters' best values for the best solver. Grid-TLSE will provide synthetic statistics from actual runs of a variety of sparse matrix solvers on chosen matrices (see Figure 1 for a typical expertise session). This process is driven by expertise scenarios, a kind of specific workflow, designed by expert to answer specific users objectives. Users may also interrogate the databases available for information and references related to sparse linear algebra. They can either submit their own problem, or use a matrix from the database available on the site including public domain matrix collections such as the Rutherford-Boeing collection[4], the University of Florida sparse matrix collection[5], etc. Second, **experts** in numerical computation and parallel/distributed programming, who are involved in writing of packages. Experts may want to compare solvers using sophisticated controls and metrics or add new solvers or new scenarios. And finally, the Grid-TLSE **manager** who will take care of users, computers, and services, matrix collections, bibliography. The manager will also need to access the current state of the grid and the list of solvers available.

One of the main difficulty with providing expertise is that solvers may take into account a lot of different control parameters. Expertise may require a large number of solver runs. The next section presents examples of some major control parameters for solving linear systems.

2.2 Sparse Direct Solvers for Linear Systems

The main service considered by Grid-TLSE aims at solving the problem $Ax = b$, with A and b as parameters and x as result. Many algorithms and software packages are available. An important

[4]http://www.cerfacs.fr/algor/Softs/RB
[5]http://www.cise.ufl.edu/research/sparse/matrices

aspect of these algorithms is that their performance (execution time, memory consumption, numerical precision, etc.) depend on the use of the structural and numerical properties of the matrix A, and of both the computer architecture (sequential, parallel, distributed) and its performance (CPU computing power, amount of memory, communication links, etc.). In order to choose the appropriate algorithm, these ones can be distinguished according to the structure of the matrix A they deal with (eg, symmetric or unsymmetric), and to the approach they use.

2.2.1 Constraints to Grid-TLSE

Tools used in the Grid-TLSE project solve $Ax = b$ when A has a sparse structure using a direct approach. The direct approach for solving $Ax = b$ consists in transforming the matrix A as a product of easier to use matrices (so called factors) and then computing the solution x using the resulting factors. Different kind of factorisation for A exist: $A = LU$, $A = QR$, $A = LL^\top$, $A = LDL^\top$, etc.

Many different software packages are likely to be integrated in Grid-TLSE. Currently, we are validating our choices with MUMPS [3], UMFPACK[13], SuperLU[21] and PaStiX [18]. It should be noted that whereas these tools take the same parameters A and b and produce the same result x, they all provide supplementary parameters in order to harness the algorithm used (also referred to as *controls*) and produce many results to qualify both the execution and the quality of the results (also referred to as *metrics*). One of the main purpose of Grid-TLSE is to help the user in choosing appropriate values for controls.

In the following subsections, we focus on the example of LU factorisation to show that even a simple numerical computation service can have many controls and many metrics that can be specific to each algorithm implementation.

2.2.2 Controls for the Factorisation Algorithm

Let's define an algorithm framework common to most solvers. In order to improve the performance of the factorisation of a matrix A, the algorithm works on a matrix of the form $PQ_RD_RAD_CQ_CP^\top$ and the linear system effectively treated is $\hat{A}\hat{x} = \hat{b}$ with $\hat{A} = PQ_RD_RAD_CQ_CP^\top$, $\hat{x} = PQ_C^\top D_C^{-1}x$ and $\hat{b} = PQ_RD_Rb$. In these equations, (i) D_R and D_C are diagonal scaling matrices for the rows and columns of A; (ii) Q_R and/or Q_C are unsymmetric permutations of A aiming at putting the large values of A onto the diagonal (one of Q_R or Q_C might be the identity); and (iii) P is a symmetric permutation whose purpose is to reduce the size of the factors during the factorisation of A.

The permutations are usually computed in the first phase of the algorithm referred to as symbolic analysis, while scalings may be performed either in the symbolic analysis phase or at the beginning of the factorisation phase. Many algorithms are available for computing permutations, for example AMD (Approximate Minimum Degree [2]), Metis (graph partitioning [20]), MMD (Multiple Minimum Degree [22]), CM (Matrix bandwidth reduction [11]). Some tools provide several of these ordering algorithms and a control in order to choose the algorithm. Depending on the tool, the permutations can either be considered as symmetric (P) or unsymmetric (either Q_R or Q_C), or as left (PQ_R) or as right ($Q_CP^\top$).

Furthermore, controls are available in order to allow a better adaptation to the properties of the matrix. Because of the tuning of the algorithms, different tools (MUMPS and UMFPACK for example) can provide different permutations for the same matrix using the same ordering algorithm (e.g., AMD).

In a second phase, the modified matrix $\hat{A}$ is factorised as an LU product. The static symmetric permutation P can then be modified into a dynamic one P_N (referred to as the numerical permutation) tuned by a pivoting threshold. The factorised problem is then $P_N\hat{A}\hat{x} = P_N\hat{b}$ (P_N should almost be the identity if one wants the estimated work and memory computed in the first phase to be reliable). Some tools hide the pivoting threshold, others provide it as a control.

The last phase referred to as the solve phase allows to compute $\hat{x}$ and then x using the L and U factors resulting from the factorisation, where other control parameters should be taken into account (iterative refinement, etc.).

The above mentioned parameters are only examples of controls; many more are available, more or less specific to each tool/solver. Parallel and distributed computing also provides sophisticated controls in order to give the best results according to the architecture and performance of the available computers.

2.3 Possible Algorithm Structures

Most of the direct algorithms for solving a sparse linear problem are composed of three phases: symbolic analysis, factorisation and solve which must be executed in sequence. It is therefore possible to share the analysis between several factorisations (with different values for the pivoting threshold) and to share a factorisation between several solves (with different values for b).

To take into account the decomposition of services into phases, different levels of granularity might be considered. At the **coarse grain** level, only the full solving service is provided. This approach is appropriate when the user does not want to share anything between several solving of the same problem with different values for algorithm control. This approach has been followed by UMFPACK early versions [12]. At the **medium grain** level, three services are available corresponding to the three phases. The user can reuse the results of some phases. However, the three services must be supplied by the same provider because they share some internal hidden data structures. This approach has been used for example in MUMPS [3]. Finally, at the **fine grain** level, phases are independent services with explicit parameters and results. The analysis is composed of several services providing orderings, scalings, etc. Services from various providers can be composed using wrappers for explicit data conversions. These wrappers may be quite complex as the explicit data structures used by each tool can be very different from one tool to the other. This approach has been followed in the development of the HSL tool family [19].

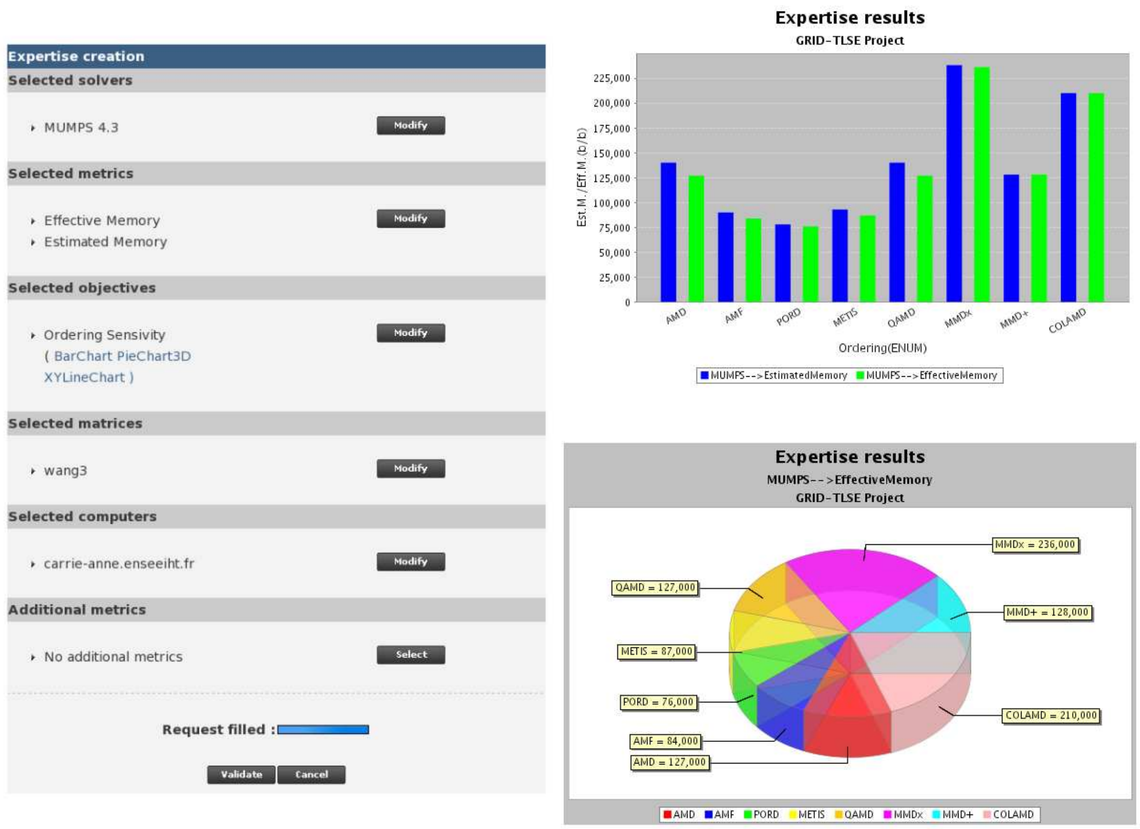
The algorithm structure description will be used in order to combine parts of different solvers and to share intermediate results between sequence of runs.

2.4 Scenarios Providing the Expertise

A typical Grid-TLSE request will ask for the behaviour of some solvers on a given matrix according to some metrics, taking into account some controls. Figure 1 presents screenshots from a typical session. The left screen is the user request providing the selected scenario/objective (**Ordering Sensitivity** which compare solvers according to the permutation algorithms); some parameters for this scenario (solver **MUMPS 4.3**, computer **carrie-anne**, matrix **wang3**); and the required metrics for graphics (**Effective Memory** and **Estimated Memory**). The right screen shows the expertise results, two synthetic graphics depending on the scenario (the dependence between the ordering and the selected metrics). This request will require a significant amount of solver runs (one run for each permutation algorithm and each matrix in order to produce the orderings, and then one run for each solver, each matrix, each computer and each ordering, see Section 4.3 and Figure 5 for more details). Sparse solver experts can usually reduce this amount: for example, some combinations of algorithms are not worth testing, or a preliminary analysis of the problem will discard some algorithmic options automatically. Depending on the user objective (eg, best *numerical accuracy*), experts usually know what type of control parameters are worth experimenting (eg, *pivoting threshold*), in which range (eg, $[0,1]$). Thus, experts can define a specific scenario in that purpose (eg, *threshold sensitivity*, see below). Furthermore, some of the results from the runs are not interesting and need to be filtered by the scenario.

The purpose of scenarios providing expertise (also referred to as expertise scenarios) defined by experts based on their knowledge and insights, is to give automatically the users precise synthetic answers while reducing the combinatorial cost of the runs. Thus, a request for expertise from the user is based on a scenario (also referred to as objective) defined by an expert.

Grid-TLSE provides a framework in order to ease the description of scenarios. Currently, the scenarios available in the prototype are:


Expertise creation

Selected solvers

- MUMPS 4.3 Modify

Selected metrics

- Effective Memory Modify
- Estimated Memory

Selected objectives

- Ordering Sensivity Modify
  (BarChart PieChart3D
  XYLineChart)

Selected matrices

- wang3 Modify

Selected computers

- carrie-anne.enseeiht.fr Modify

Additional metrics

- No additional metrics Select

Request filled :

Validate Cancel

Figure 1: Expertise session : query and result screenshots

1. **Solve**: to use a range of solvers with their default control parameters, and report the values of some metrics,

2. **Threshold Sensitivity**: to study the quality of the numerical solution as a function of the pivoting threshold control,

3. **Ordering Sensitivity**: to evaluate the behaviour of solvers according to the ordering heuristic control (see Figure 5),

4. **Minimum Time**: to estimate which combination of solver/ordering leads to the smallest computation time on a given problem (see Figure 6).

2.5 Why Do We Need a Grid ?

The previous sections have shown that solvers depend on a huge number of controls in order to harness the algorithm (both numerical and performance aspects). For example, MUMPS, SuperLU and UMFPACK provide around 100, 20, and 20 controls respectively. These controls can take a large number of values. Most of the scenarios consist in helping users in the choice of values for these parameters. Therefore, these scenarios will require a huge number of solver runs. Using the previous three solvers on a given problem, the *Ordering sensitivity* scenario mentioned above will produce around 39 runs. Grid-TLSE is clearly a multi-parametric application and grid has been shown to be well adapted to this kind of problems.

Moreover, users can have very different requirements in terms of solvers, libraries, operating systems, computer architecture and performance. Grid-TLSE therefore needs to be able to access to as many different systems as possible. Public solvers can usually be installed on most of the available computers. This is not the case for private ones. For example, industrial partners may wish to give access to one of their solvers on one of their own computers. Grid-TLSE then needs to access software only available on some private remote site. Users may provide very huge or security sensitive problems which should not be communicated to other computers. Then, Grid-TLSE needs to run solvers on the sites where these problems are available.

Furthermore, in order to use the computational resources efficiently, we require a grid scheduling middleware that will provide an easy access to solvers running somewhere on the grid and improve the use of the computing power available for Grid-TLSE with sophisticated scheduling algorithms.

3 DIET: Distributed Interactive Engineering Toolbox

3.1 Architecture

The aim of the DIET project is to provide a toolbox that will allow different applications to be ported efficiently over the grid and to allow our research team to validate theoretical results on scheduling or on high performance data management for heterogeneous platforms.

The DIET architecture is based on a hierarchical approach to provide scalability. The architecture is flexible and can be adapted to diverse environments including heterogeneous network hierarchies. DIET is implemented in Corba and thus benefits from the many standardised, stable services provided by freely-available and performant Corba implementations. DIET is based on several components. A **Client** is an application that uses DIET to solve problems using an RPC approach. Users can access DIET via different kinds of client interfaces: web portals, PSEs such as Scilab, or from programs written in C or C++. A **SeD**, or server daemon, provides the interface to computational servers and can offer any number of application specific computational services. A SeD can serve as the interface and execution mechanism for a stand-alone interactive machine, or it can serve as the interface to a parallel supercomputer by providing submission services to a batch scheduler. **Agents** provide higher-level services such as scheduling and data management. These services are made scalable by distributing them across a hierarchy of agents composed of a single **Master Agent (MA)**, several **Agents (A)**, and **Local Agents (LA)**. Figure 2 shows an example of a DIET hierarchy.

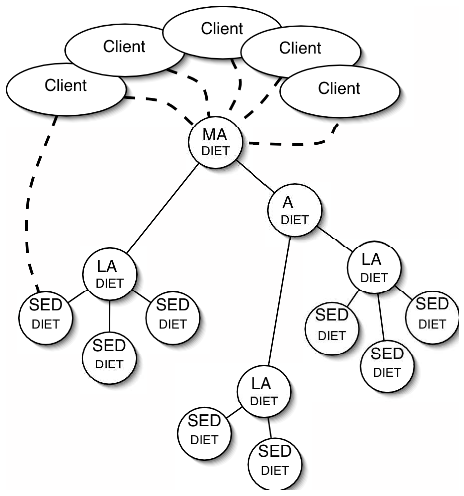


Figure 2: DIET hierarchical organisation.

A **Master Agent** is the entry point of our environment. Clients submit requests for a specific computational service to the MA. The MA then forwards the request in the DIET hierarchy and the child agents, if any exist, forward the request onwards until the request reaches the SeDs. The

SeDs then evaluate their own capacity to perform the requested service; capacity can be measured in a variety of ways including an application-specific performance prediction, general server load, or local availability of data-sets specifically needed by the application (see next section). The SeDs forward their responses back up the agent hierarchy. The agents perform a distributed collation and reduction of server responses until finally the MA returns to the client a list of possible server choices sorted in order of desirability. The client program may then submit the request directly to any of the proposed servers, though typically the first server will be preferred as it is predicted to be the most appropriate server.

Several such hierarchies can be connected either directly or in a peer-to-peer fashion. More information about the behavior of DIET can be found in [9].

3.2 Performance Evaluation

Scheduling tasks on computers comes down to mapping task requirements to system availability. Requirements of routines group principally the time and the memory space necessary to their execution, as well as the amount of generated communication. These values depend naturally on the chosen implementation and on input parameters of the routine, but also on the machine on which the execution takes place. System availability information captures the number of the machines and their speed, as well as their status (down, available, or allocated through a batch system). One must also know the topology, the capacity, and the protocols of the network connecting these machines. From the scheduling point of view, the actual availability and performance of these resources is more important than their previous use or the theoretical peak performance.

The goal of FAST [26] is to constitute a simple and consistent Software Development Kit (SDK) for providing client applications with accurate information about task requirements and system performance information, regardless of how theses values are obtained. The library is optimised to reduce its response time, and to allow its use in an interactive environment. It is based on NWS (Network Weather Service) [29]. At FAST install time, a list of problems of interest are specified along with their interfaces; FAST then automatically performs a series of macro-benchmarks which are stored in a database for use in the DIET scheduling process. For some applications, a suite of automatic macro-benchmarks can not adequately capture application performance. In these cases, DIET also allows the server developer to specify an application-specific performance model to be used by the SeD during scheduling to predict performance. Although the primary targeted application class consists of sequential tasks, this approach has been successfully extended to address parallel routines as well, as explained in more details in [16].

On the other hand, the performance evaluation of sparse direct solvers (and of sparse linear algebra tools in general) is a challenging problem. As discussed in Section 2, the performance of a solver is not known in advance and depends on many parameters. In fact this is precisely one of the main reasons why an expertise site such as Grid-TLSE is being developed: instead of having an experienced user conducting a range of experiments (and installing the associated software) to discover which solver with which set of controls is the most appropriate for his/her given problem, most of this process is automatized, following scenarios (see Sections 2.4 and 4.3) that can be defined by experts from the field. In this context, statistics (also referred to as metrics) on the performance, number of floating-point operations, memory consumption, efficiency of the computation, are indeed one of the results from the expertise and are only known *after* the execution of a particular solver. Having said that, some sort of estimate of the time and memory usage of a sparse direct solver on a given problem is very useful in order to provide more information to the middleware layer in charge of dispatching jobs onto the grid resources. This information helps scheduling the jobs in an efficient manner, both in the case of the Grid-TLSE application, or more generally, in the case of a grid service dedicated to the solution of sparse systems of equations (imagine a client who does not have the hardware and/or software resources locally for that part of his computations). Although we cannot predict in advance the performance of a solver on a problem, we give below some partial solutions:

Extrapolation or data-mining from previous results. First, consider a given application where *similar* sparse matrices arise depending on a problem size and we suppose that the various input parameters to a given solver are fixed. There is a good chance that the computational cost (time, memory) of the solver will increase smoothly with the problem size. It is thus possible to bench the solver for several problem sizes and extrapolate the results to form a polynomial approximation of the cost function; in such a context, FAST can be used. If we now consider the more general case, where a new problem is to be solved with a new set of solver parameters, this no more applies. The main difficulty here is to define important parameters of the matrix versus application domain, that helps predicting the costs. Furthermore, the type of solver and the solver's parameters also have a strong impact. Since all the statistics from expertise runs are stored, data-mining techniques should be used (once Grid-TLSE is in production mode and enought statistics are available).

Allow for the cost of an analysis phase. As seen in Section 2.3, sparse direct solvers generally work in three distinct phases. The first one, *symbolic analysis*, analyses the graph of the considered sparse matrix and performs the so-called *symbolic factorisation*, before actual computations are performed in the subsequent factorisation and solution phases. Depending on scales considered, the cost of an analysis phase may be affordable in the performance prediction itself, although this already requires the complete structure of the matrix to be transferred on the server chosen in that purpose. Indeed, this step can be implemented as a normal service from the middleware itself, whose cost and resource requirements are easier to evaluate. Then the analysis phase provides very useful information regarding the computational work of the next phases: number of floating-point operations and estimate of memory consumption; such information can be combined with information on the considered solver and the status of the servers to predict the performance. Also, in the parallel case a function providing an estimate of the parallel efficiency with respect to the number of processors can be used.

As shown above these solutions are not immediate (and have not been implemented) and what we plan to to start with in practice is something much less accurate that provides a rough idea of what the computational cost will be. This could simply be based on the size of the matrix, degree of connectivity of the graph, and a simple function providing an upperbound of the cost, given an average Mflops rate per solver.

Then this approach can be enhanced by using ideas from the two approaches above: for example, perform the symbolic analysis phase on a simplified matrix (by compressing the graph) to get an idea of the computational costs on that simplified case, and study how this new parameter combined with characteristics from the original problem can be used to build a more accurate cost function. In any case, studying the behaviour of TLSE once lots of experiments are submitted to the site will be the basis to improve the performance prediction in an incremental manner.

3.3 Data Management

GridRPC environments such as NetSolve, Ninf, and DIET are based on the client-server programming paradigm. However, generally in this paradigm, no data management is performed. Like in the standard RPC model, request parameters (input and output data) are sent back and forth between the client and the remote server. A data is not supposed to be available on a server for another step of the algorithm (a new RPC) once a step is finished. This drawback can lead to extra overhead due to useless communications over the net.

A first data management service has been developed for the DIET platform [15] called Data Tree Manager (DTM). This DIET data management model is based on two key elements: the data identifiers and the Data Tree Manager (DTM). To avoid multiple transmissions of the same data from a client to a server, the DTM allows to leave data inside the platform after computation while data identifiers will be used further by the client to reference its data. This approach is well adapted to sharing intermediate results between linear algebra solvers (see Section 2.3) in an experiment plan (see Section 4.1).

The second approach consists in using JuxMem (Juxtaposed Memory) [4] which is a peer-to-peer architecture which provides memory sharing service allowing peers to share memory data, and not only files. It is described in an other chapter of this book. More information about the data management can be found in [9].

3.4 Deployment

This section focus on the deployment of DIET. Although the deployment of such an architecture may be constrained e.g., firewall, right access or security, its efficiency heavily depends on the quality of the mapping between its different components and the grid resources. In [8] we have proposed a new model based on linear programming to estimate the performance of a deployment of a hierarchical PSE. The advantages of our modelling approach are: evaluate a virtual deployment before a real deployment, provide a decision builder tool (i.e., designed to compare different architectures or add new resources) and take into account the platform scalability. Using our model, it is possible to determine the bottleneck of the platform and thus to know whether a given deployment can be improved or not.

In complementary work of the previous theoretical approach we developed GoDIET, a new tool for the configuration, launch, and management of DIET on computational grids. GoDIET users write an XML file describing their available compute and storage resources and the desired overlay of DIET 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 allows shutdown of all launched software.

The following associated services may be used in conjunction with DIET. **LogService.** LogService is a CORBA-based logging service. This software package provides interfaces for generation and sending of log messages by distributed components, a centralised service that collects and organises all log messages, and the ability to connect any number of listening tools to whom LogService will send all or a filtered set of log messages. **VizDIET.** VizDIET is a tool that provides a graphical view of the DIET deployment and detailed statistical analysis of a variety of platform characteristics such as the performance of request scheduling and solves. To provide real-time analysis and monitoring of a running DIET platform, VizDIET can register as a listener to LogService and thus receives all platform updates as log messages sent via CORBA. Alternatively, to perform visualisation and processing post-mortem, VizDIET uses a static log message file that is generated during run-time by LogService and set aside for later analysis. This approach will also be used by Grid-TLSE in order to provide the project manager with insights on the state of the grid and the running solvers. Figure 3 provides an overview of the interactions between a running DIET platform, LogService, and VizDIET.

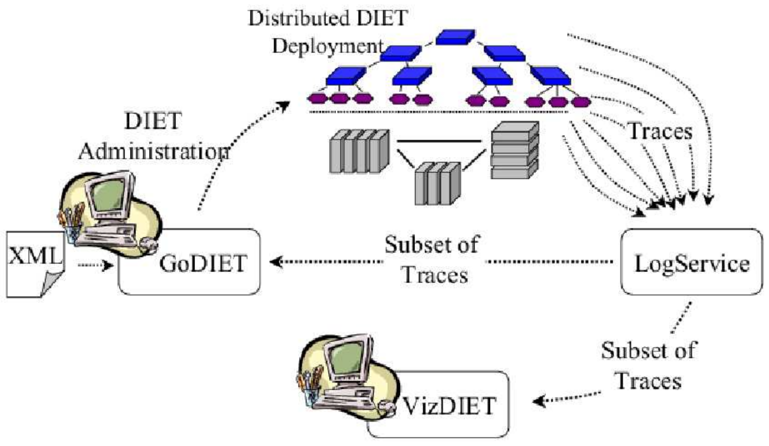


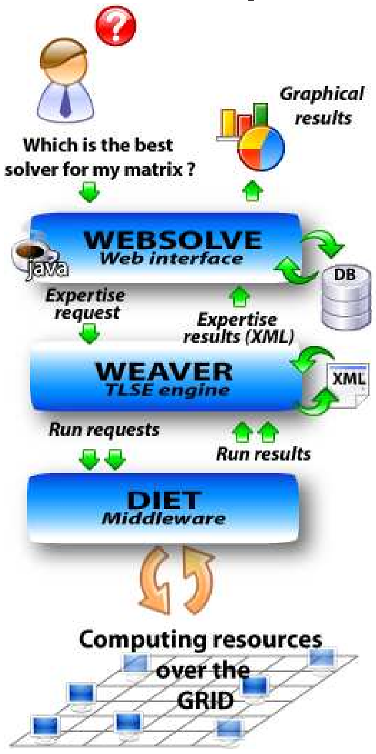
Figure 3: Interaction of GoDIET, LogService, and VizDIET to assist users in controlling and understanding DIET platforms.

4 TLSE Architecture

The previous sections have shown Grid-TLSE main purpose, its need for a grid and the DIET middleware to harness it.

One major point is that Grid-TLSE is not a simple static web portal providing predefined services such as NEOS which provides a specific interface for each kind of tools. One main requirement is that: "*it should be easy to add solvers and scenarios. New scenarios should be able to use old solvers. Old scenarios should be able to use new solvers.*". For this purpose, Grid-TLSE is a dynamic framework for building expertise providing web site which eases the description of scenarios and of all the data required for scenarios (solvers, computers, services, expertise scenarios, problems, etc.). The description of the structure of data are referred to as meta-data (see Section 4.2).

This section now presents the Grid-TLSE internal architecture and its interaction with DIET.



The main components of the Grid-TLSE site are the followings. **WebSolve** allows a user using a standard WWW navigator to submit requests for computation or expertise to a grid, browse the matrix database, upload/download a matrix, monitor the submitted requests, manage and add solvers and scenarios, and finally check for their correct execution. Most of the Web interface is dynamic: it is built according to the meta-data (see Figure 1). **Weaver** converts a general request for expertise into sequences of elementary solver runs (see Figure 4 in Section 4.1). It is also in charge of the deployment and the exploitation of services over a grid through the DIET middleware. The expertise providing kernel is fully dynamic in the same sense as WebSolve, all the services rely on the meta-data. **DIET** provides an access to solvers and data. Finally, the **Database** stores the required data for the whole project. In particular, it contains all the meta-data.

One of the main research issue in Grid-TLSE is the specification of the procedures for providing expertise and the management of the site. In particular, we have designed a framework for the description and management of solvers and scenarios, and have developed procedures for: adding new software packages, graphical definition of new scenarios, exploiting the computed statistics i.e. being able to "reuse" results, avoid repetition of runs, and study the typical behaviour of a solver or the properties of a matrix.

4.1 Running an Expertise Providing Session

Figure 4 illustrates the various steps of an expertise providing session. First, the user interacts with the WebSolve interface in order to choose an expertise scenario (the objective of the session) and provide the appropriate parameters for this scenario (see the left screen of Figure 1). These parameters are described by the meta-data which defines the selected scenario (see Figures 5 and 6). WebSolve interactively checks that the parameter values are valid according to the meta-data. Then, this request is forwarded to the Weaver kernel. According to the description of the scenario (see Figures 5 and 6), Weaver builds one or more expertise steps (which correspond to execution

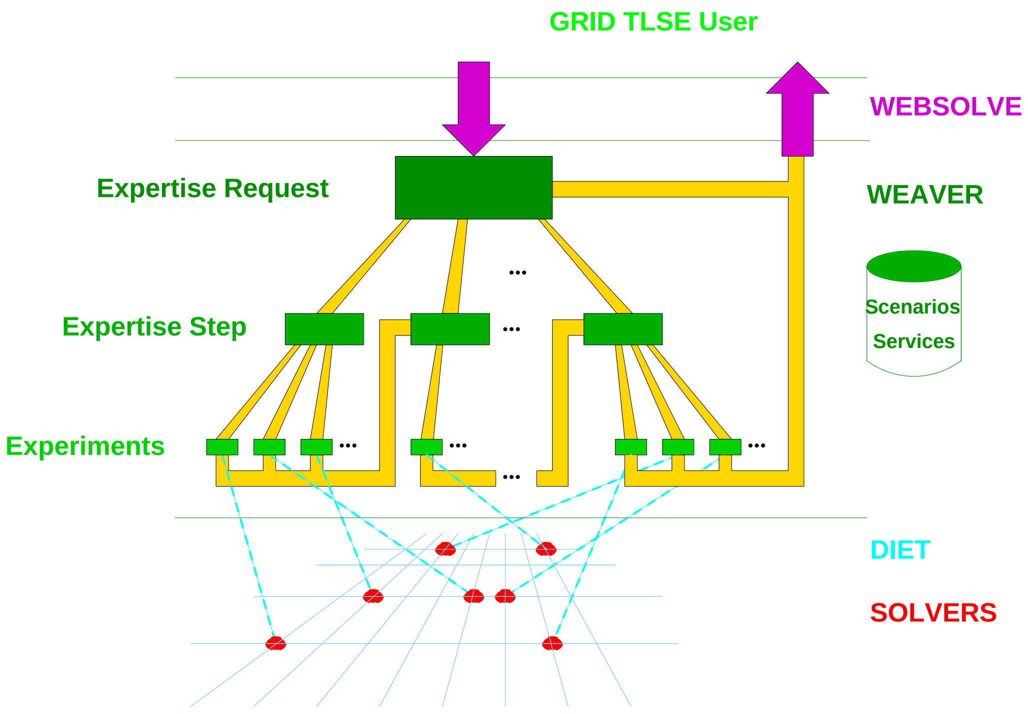


Figure 4: Whole expertise providing session

operators in the scenarios). Each expertise step produces an experiment plan. An experiment is a partially valued set of features which represents a solver run. Running an experiment will forward this set to the appropriate solver on the grid thought DIET which will send back the fully valued experiment resulting from the solvers run. All the results of an experiment plan are processed according to the scenario in order to produce the next expertise step. And finally, the results of the last experiment plan are forwarded to WebSolve which stores all the raw results and then produces synthetic graphics according to the scenario and the user request (see the right screen of Figure 1).

The number of expertise steps is therefore dynamic and depends on the results of the experiments. The expertise process terminates as the only iterative operators in a scenario are, on one hand a *foreach* applied to finite sets of values, and on the other hand, recursive traversal of finite static trees used in the meta-data. The scenario is therefore a kind of dynamic workflow whose execution depends on the intermediate results.

DIET is used in order, on one hand, to schedule and execute experiments on the most adapted available solvers on the grid, and on the other hand, to share intermediate results inside an experiment plan or between various experiment plans (see Section 3.3). Scenarios express data-flow dependencies which are used by the DIET persistence facilities in order to reduce the communication costs by an appropriate scheduling.

4.2 Grid-TLSE Meta-Data Framework for Sparse Solvers

Sparse direct solvers are quite similar since they all solve a sparse linear system using some computer resources and provide a result with a given numerical precision. But they are also very different in practice since they all use different algorithms with their own controls as shown in Section 2.2.

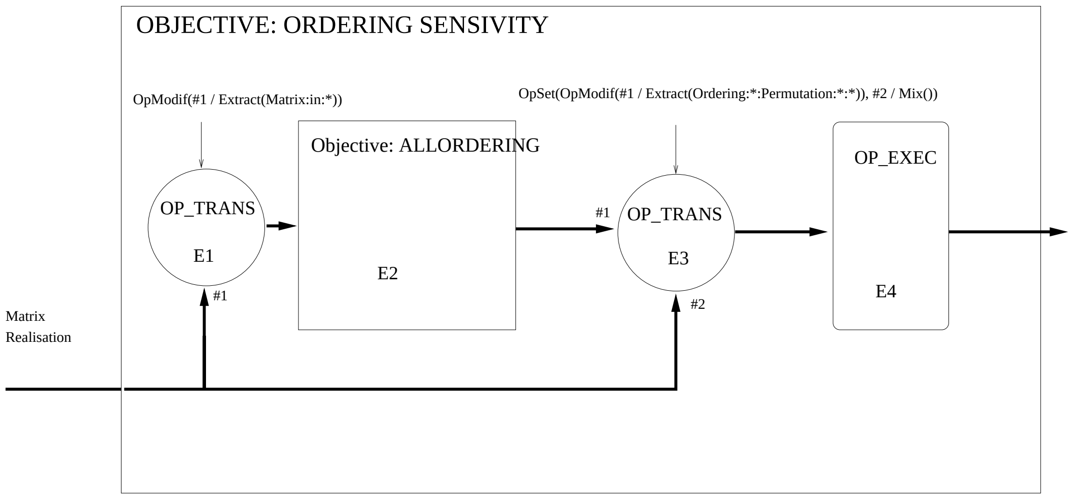

OBJECTIVE: ORDERING SENSIVITY
OpModif(#1 / Extract(Matrix:in:*))
OpSet(OpModif(#1 / Extract(Ordering:*:Permutation:*:*)), #2 / Mix())
Objective: ALLORDERING
OP_EXEC
OP_TRANS
E1
#1
E2
#1
OP_TRANS
E3
#2
E4
Matrix
Realisation

Figure 5: Ordering sensitivity

The main purpose of the project is to help the user in choosing the right solver and an appropriate selection of control values, Grid-TLSE must then be able to proceed to a comparison of the solvers. As a consequence, all solvers must provide a similar interface to the scenarios, and all scenarios must provide a similar interface to the client. One simple approach is to write wrappers around all the solvers that take the same parameters and produce the same results. This can be quite heavy in term of development cost and is very sensitive to the addition of new parameters that may require the modification of all wrappers.

A more interesting approach relies on the use of meta-data which define all the possible parameters and results (and their possible values) for each solver (see [24, 25] for a description of Grid-TLSE meta-data framework). Given a set of solvers, the intersection of their parameter's set (and the parameter's values) will offer more possibilities than the common interface approach. The meta-data approach allows the expert to define all the possible parameters and to describe each solver according to these parameters. The scenario approach allows the expert to define a scenario using the meta-data and to define the graphics returned back to the user as a result of the request for expertise. The experiments are then represented as a set of feature (meta-data values) as seen in the previous section. This set will be transmitted through the DIET middleware to a minimal wrapper translating meta-data values to real solver parameters (for example, internal ordering of type AMD will translate to *ICNTL(7)=0* for MUMPS). On return, the statistics from the solver, or actual performance measurements, are translated back into metrics transmitted to Weaver.

4.3 Expertise Scenarios

We now give some details about the "Ordering sensitivity" scenario to illustrate its hierarchical structure (see Figure 5). This scenario requires the user to provide the *matrix* and *realisation* parameters (see the left screen of Figure 1). **Phase 1 (sub-scenario AllOrdering)** executes the sub-scenario which has the following effect: if only one solver has been specified by the user, run the solver to get all its internal orderings; if more than one solver has been specified, run the solvers in order to get all possible orderings. **Phase 2 (OP_EXEC execution operator, an expertise step which produces experiments)** runs the solvers in order to obtain the values of the required metrics for each ordering. For metrics of type "estimated": only the analysis is performed for each required solver, for metrics of type "effective": the factorisation is also performed. The scenario then report metrics for all combinations of solvers/orderings.

This scenario is static: the number and kind of experiment plans do not depend on the results

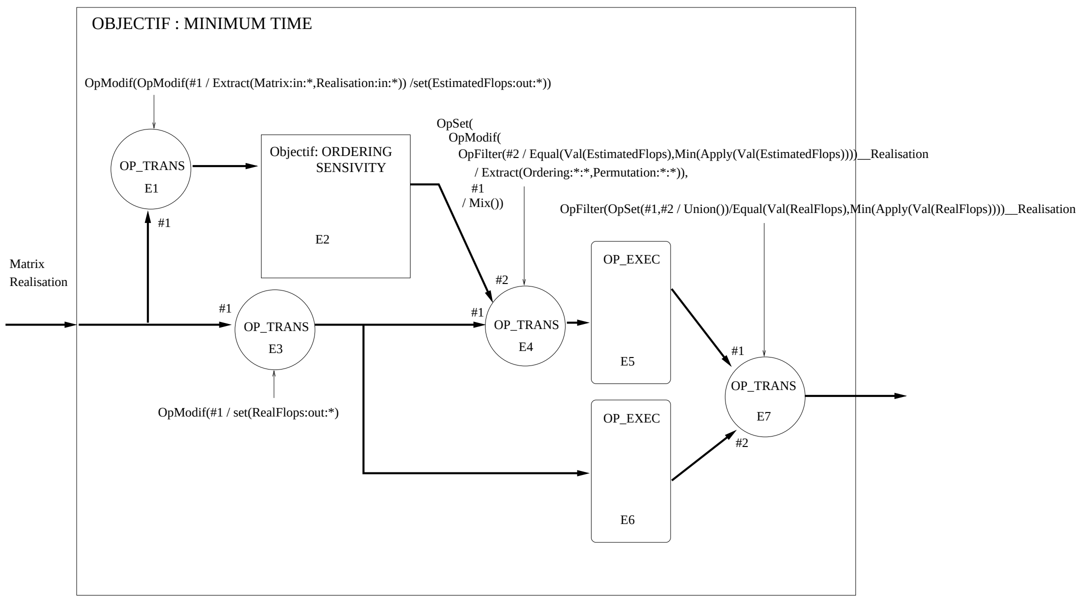

OBJECTIF : MINIMUM TIME
OpModif(OpModif(#1 / Extract(Matrix:in:*,Realisation:in:*)) /set(EstimatedFlops:out:*))
OP_TRANS
E1
#1
Objectif: ORDERING SENSIVITY
E2
OpSet(
OpModif(
OpFilter(#2 / Equal(Val(EstimatedFlops),Min(Apply(Val(EstimatedFlops))))__Realisation
/ Extract(Ordering:*:*,Permutation:*:*)),
#1
/ Mix())
OpFilter(OpSet(#1,#2 / Union())/Equal(Val(RealFlops),Min(Apply(Val(RealFlops))))__Realisation
Matrix
Realisation
#1
OP_TRANS
E3
OpModif(#1 / set(RealFlops:out:*)
#2
#1
OP_TRANS
E4
OP_EXEC
E5
OP_EXEC
E6
#1
OP_TRANS
E7
#2

Figure 6: Minimum time

of the experiments but only on the values of the various meta-data (number of permutation algorithms, of solvers, of matrices, of computers, . . .). Scenarios can also be dynamic such as the **Minimum Time** (see Figure 6) scenario which uses the **Ordering sensitivity** scenario to produce all potential pair of ordering/solver and compute the estimated execution cost through a low cost symbolic analysis. It then selects the best ordering for each solver and produces a new experiment plan in order to compute the effective execution time and report the user the best ordering/solver pair for its problem.

It should be noted that expertise providing scenarios are some special kind of workflow. This point will be further explored in the future.

5 Related Work

Several other Network Enabled Server systems have been developed in the past [5, 17]. Among them, NetSolve [6] and Ninf [23] have pushed further the research around the GridRPC paradigm.

NetSolve [6] has been developed at the University of Tennessee, Knoxville. NetSolve allows the connection of clients (written in C and support interface for another languages such as C++, Fortran, Matlab, etc.) to solve requests sent to servers found by a single agent. This centralized agent maintains a list of available servers along with their capabilities. Servers sent at a given frequency information about their status. Scheduling is done based on simple models provided by the application developers, LINPACK benchmarks executed on remote servers, and information given by NWS. Some fault tolerance is also provided at the agent level. Data management is also done either through request sequencing or using IBP. Security is also addressed using Kerberos. Client Proxies ensure a good portability and interoperability with other systems like Ninf or Globus [7].

Ninf [7] is a NES system developed at the Grid Technology Research Center, AIST in Tsukuba. Close to NetSolve in its initial design choices, it has evolved towards several interesting approaches using either Globus or Web Services [28]. The performance of the platform can be studied using a powerful tool called BRICKS.

[6]http://www.cs.utk.edu/netsolve
[7]http://ninf.apgrid.org/

The main differences between the NES systems presented in this section and DIET are mainly the use of distributed scheduling for DIET that allows a better scalability when the number of clients is large and the request frequency is high, the data-management facilities, the possibility of adapting the schedulers for a specific application, and the use of CORBA as a middleware [9].

6 Conclusion and Future Work

In this paper, we presented the overall architecture of DIET, a scalable environment for the deployment of applications based on the Network Enabled Server paradigm on the grid. As NetSolve and Ninf, DIET provides an interface to the GridRPC API defined within the Global Grid Forum.

Our main objective is to improve the scalability of the platform using a distributed set of agents managing a large set of servers available through the network. The dynamic change in the number of schedulers allows to ensure a level of performance adapted to the characteristics of the platform (number of clients, number and frequency of requests, performance of the target platform). Data management is also an important part of the performance gain when dependences exist between requests. The management of the platform is handled by several tools like GoDIET for the automatic deployment of the different components, LogService for the monitoring, and VizDIET for the visualisation of the behaviour of the DIET's internals. Many applications have been ported on DIET around chemical engineering, physics, bioinformatic, robotic, etc. More information is given on our web (http://graal.ens-lyon.fr/DIET/). Our future work will consist in adding more flexibility using plugin schedulers, improving the dynamicity of the platform using P2P connection (with JXTA), improving the relations between the schedulers and the data managers, and finally to validate the whole platform at a large scale within the GRID5000 project (`http://www.grid5000.org`). We also investigate the use of Grid services within DIET.

We presented one application around sparse linear solvers. Through a web site, Grid-TLSE provides help for end-users who want to select the most appropriate solver for their problems, and a testbed for expert users who want to compare solvers. The Grid-TLSE framework relies on keyword-like meta-data in order to describe the linear algebra services. This only allows basic grid service trading based on the service name. We are currently designing a more sophisticated description based on mathematical properties of linear algebra operators. The user will then give a mathematical description of the required services and the framework will look for an available service or a composition of available services in order to satisfy the user requirements. When the provided service is a composition, the framework will interact with DIET in order to provide the best composition according to the state of the grid. Another point is that the Grid-TLSE is currently applied only to direct solvers. Our future work consists in extending it to the specificities of iterative solvers.

The combination of these two projects, resulting in the Grid-TLSE expert site, should be very useful to users from various applications areas. Furthermore, we believe that the simplicity to experiment new combinations of algorithms, as well as the large amount of statistics available on the site will provide new insights to developers of sparse solvers. This will help them to extend their understanding of the field and improve their algorithms.

Acknowledgments

This work was supported in part by the ACI GRID (ASP and TLSE) and the RNTL (GASP) of the French National Fund for Science.

References

[1] P. Amestoy and M. Pantel. Grid-TLSE: A web expertise site for sparse linear algebra. In *Sparse Days and Grid Computing in St Girons*, june 2003.

http://www.cerfacs.fr/algor/PastWorkshops/SparseDays2003.

[2] P. R. Amestoy, T. A. Davis, and I. S. Duff. An approximate minimum degree ordering algorithm. *SIAM Journal on Matrix Analysis and Applications*, 17:886–905, 1996.

[3] P. R. Amestoy, I. S. Duff, and J.-Y. L'Excellent. Multifrontal parallel distributed symmetric and unsymmetric solvers. *Comput. Methods Appl. Mech. Eng.*, 184:501–520, 2000.

[4] G. Antoniu, L. Bougé, and M. Jan. JuxMem: An adaptive supportive platform for data sharing on the grid. In *Proceedings Workshop on Adaptive Grid Middleware (AGRIDM 2003)*, pages 49–59, 2003.

[5] P. Arbenz, W. Gander, and J. Mori. The Remote Computational System. *Parallel Computing*, 23(10):1421–1428, 1997.

[6] D. Arnold, S. Agrawal, S. Blackford, J. Dongarra, M. Miller, K. Sagi, Z. Shi, and S. Vadhiyar. Users' Guide to NetSolve V1.4. Computer Science Dept. Technical Report CS-01-467, University of Tennessee, Knoxville, TN, July 2001.

[7] D.C. Arnold, H. Casanova, and J. Dongarra. Innovations of the NetSolve Grid Computing System. *Concurrency And Computation: Practice And Experience*, 14:1–23, 2002.

[8] E. Caron, P.K. Chouhan, and A. Legrand. Automatic Deployment for Hierarchical Network Enabled Server. In *The 13th Heterogeneous Computing Workshop*, 2004.

[9] E. Caron and F. Desprez. DIET: A Scalable Toolbox to Build Network Enabled Servers on the Grid. Technical Report 2005-23, LIP ENS Lyon, 2005.

[10] E. Caron, F. Desprez, F. Lombard, J.-M. Nicod, M. Quinson, and F. Suter. A Scalable Approach to Network Enabled Servers. In *Proc. of EuroPar 2002*, 2002.

[11] E. Cuthill. Several strategies for reducing the bandwidth of matrices. In D. J. Rose and R. A. Willoughby, editors, *Sparse Matrices and Their Applications*, New York, 1972. Plenum Press.

[12] T. A. Davis and I. S. Duff. A combined unifrontal/multifrontal method for unsymmetric sparse matrices. *ACM Trans. on Mathematical Soft.*, 25(1):1–19, 1999.

[13] T.A. Davis. Algorithm 832: Umfpack v4.3—an unsymmetric-pattern multifrontal method. *ACM Trans. Math. Softw.*, 30(2):196–199, 2004.

[14] M. Daydé, L. Giraud, M. Hernandez, J.-Y. L'Excellent, M. Pantel, and C. Puglisi. An overview of the Grid-TLSE project. In *Proceedings of 6th International Meeting VECPAR'04, Valencia, Spain*, pages pp 851–856, June 2004.

[15] B. Del Fabbro, D. Laiymani, J.-M. Nicod, and L. Philippe. Data management in grid applications providers. In *IEEE Int. Conf. DFMA'05*, February 2005.

[16] F. Desprez, M. Quinson, and F. Suter. Dynamic Performance Forecasting for Network Enabled Servers in a Metacomputing Environment. In *Procs of the Int. Conf. on Parallel and Distributed Processing Techniques and Applications*, 2001.

[17] M.C. Ferris, M.P. Mesnier, and J.J. Mori. NEOS and Condor: Solving Optimization Problems Over the Internet. *ACM Trans. on Mathematical Sofware*, 26(1):1–18, 2000.

[18] P. Hénon, P. Ramet, and J. Roman. PaStiX: A High-Performance Parallel Direct Solver for Sparse Symmetric Definite Systems. *Parallel Computing*, 28(2):301–321, January 2002.

[19] HSL. A collection of Fortran codes for large scale scientific computation, 2000.

[20] G. Karypis and V. Kumar. MeTiS – *A Software Package for Partitioning Unstructured Graphs, Partitioning Meshes, and Computing Fill-Reducing Orderings of Sparse Matrices – Version 4.0.* University of Minnesota, September 1998.

[21] X. S. Li and J. W. Demmel. SuperLU_DIST: A scalable distributed-memory sparse direct solver for unsymmetric linear systems. *ACM Trans. on Mathematical Soft.*, 29(2), 2003.

[22] J. W. H. Liu. Modification of the minimum degree algorithm by multiple elimination. *ACM Trans. on Mathematical Soft.*, 11(2):141–153, 1985.

[23] H. Nakada, M. Sato, and S. Sekiguchi. Design and Implementations of Ninf: towards a Global Computing Infrastructure. *Future Generation Computing Systems, Metacomputing Issue*, 15(5-6):649–658, 1999. http://ninf.apgrid.org/papers/papers.shtml.

[24] M. Pantel, C. Puglisi, and P. Amestoy. Test for large scale systems of equations: meta-data for solvers, matrices and computers. In *PMAA'04*, 2004.

[25] Marc Pantel, Chiara Puglisi, and Patrick Amestoy. Grid, components and scientific computing. Technical Report TR/TLSE/05/07, ENSEEIHT-IRIT, 2005.

[26] M. Quinson. Dynamic Performance Forecasting for Network-Enabled Servers in a Metacomputing Environment. In *Int. Workshop on Performance Modeling, Evaluation, and Opt. of Parallel and Dist. Syst. (PMEO-PDS'02), in conjunction with IPDPS'02*, 2002.

[27] K. Seymour, C. Lee, F. Desprez, H. Nakada, and Y. Tanaka. The End-User and Middleware APIs for GridRPC. In *Workshop on Grid Application Programming Interfaces, In conjunction with GGF12*, Brussels, Belgium, September 2004.

[28] S. Shirasuna, H. Nakada, S. Matsuoka, and S. Sekiguchi. Evaluating Web Services Based Implementations of GridRPC. In *Proceedings of the 11th IEEE International Symposium on High Performance Distributed Computing (HPDC-11 2002)*, pages 237–245, 2002.

[29] R. Wolski, N. T. Spring, and J. Hayes. The Network Weather Service: A Distributed Resource Performance Forecasting Service for Metacomputing. *Future Generation Computing Systems, Metacomputing Issue*, 15(5–6):757–768, Oct. 1999.