Tunable parallel experiments in a GridRPC framework: application to linear solvers

Y. Caniou[1†‡], J.-S. Gay[2†‡], and P. Ramet[3†§]

[1] LIP-ÉNS de Lyon, Université Claude Bernard de Lyon (yves.caniou@ens-lyon.fr)
[2] LIP-ÉNS de Lyon (Jean-Sebastien.Gay@ens-lyon.fr)
[3] LABRI, Université Bordeaux 1 (ramet@labri.fr)

Abstract. The use of scientific computing centers becomes more and more difficult on modern parallel architectures. Users must face a large variety of batch systems (with their own specific syntax) and have to set many parameters to tune their applications (*e.g.*, processors and/or threads mapping, memory resource constraints). Moreover, finding the optimal performance is not the only criteria when a pool of jobs is submitted on the Grid (for numerical parametric analysis for instance) and one must focus on the wall-time completion. In this work we tackle the problem by using the DIET Grid middleware that integrates an adaptable PASTIX service to solve a set of experiments issued from the simulations of the ASTER project.
Key words: Grid computing, Sparse linear solver, Performance prediction, Application specific plug-in scheduling

1 Introduction

Parallel computing and the design of high-performance codes to be executed on today's computing platforms are one of the major research activities in computer science. Such architectures are now parallel computers organized as a large network of SMP nodes and/or Grid platforms. On an other hand, solving large sparse systems of linear equations is a crucial and time-consuming step, arising in many scientific and engineering applications. Consequently, many parallel techniques for sparse matrix factorization have been studied and implemented.

In the context of the ASTER project (*Adaptive MHD Simulation of Tokamak ELMs for* ITER), we develop and implement methods to improve the simulation of MHD instabilities that are needed to evaluate mechanisms to control the energy losses observed in the standard tokamak operating scenario (ITER). To resolve a wide range of timescales, a fully implicit time evolution scheme is used; this leads to a large sparse matrix system to be solved at every time step. To reduce the large memory requirements of the sparse matrix solve, the PASTIX library [7] is being extended to support preconditioners using incomplete factorization.

† This work is supported by the REDIMPS project JST-CNRS
‡ This work is supported by the LEGO project ANR-05-CIGC-11
§ This work is supported by the ASTER project ANR-06-CIS-1 and SOLSTICE project ANR-06-CIS-10

One of the aim of the ASTER project is to define the choice of optimal parameters of the solver on a collection of test cases and to analyze the efficiency and convergence for both parallel and sequential implementations [4]. Then, we must perform an exhaustive set of experiments, whose objective is to collect the benchmark results induced by this parametric analysis.

We address the efficiency problem with a Grid architecture relying on the DIET [3] Grid middleware. The proposed solution integrates the development of a DIET client/server which gives access to the PASTIX service over the Grid, a transparent batch parallel job submission mechanism to address batch systems heterogeneity as well as mechanisms to obtain static *and* dynamic information on the resources of a site, a parallel job tuning to address the moldability of PASTIX (the possibility to set the number of processors to use at launch time), and a distributed application-specific scheduler.

2 Related Work

The TLSE project [1] (Test for Large Systems of Equations) aims to provide an expert Grid system, particularly to evaluate sparse direct solvers. TLSE relies on the DIET Grid middleware to submit the computing analysis on the Grid. The context of work is different than the one in this paper, because submission of parallel jobs was not available within the DIET API and such had to be done case by case (forks or batch scripts had to be hard coded if used), and iterative solvers functionalities (incomplete factorization for instance) cannot be taken into account. In any case, the platform will not allow performance predictions and approximate wall-time. Furthermore, we want to take advantage of the moldability of PASTIX parallel jobs, which can be tuned accordingly, for example to benefit of the maximum idle resources on a site.

3 Presentation of PASTIX

PASTIX [2] is a scientific library that provides a high performance parallel solver for very large sparse linear systems based on block direct and block ILU(k) iterative methods.

The PASTIX library uses the graph partitioning and sparse matrix block ordering package `Scotch` [8]. PASTIX is based on an efficient static scheduling and memory manager by taking into account very precisely the computational costs of the BLAS primitives, the communication costs and the cost of local aggregations.

In the context of SMP node architectures, we fully exploit shared memory advantages. A relevant approach is then to use an hybrid MPI-thread implementation. This not yet explored approach in the framework of direct solver aims at efficiently solving 3D problems with much more than 10 millions of unknowns. We have shown that this approach allows a great reduction of the memory required for communications [6]. Hybrid MPI-thread batch scheduling is then crucial to solve large problems even if those requirements are often difficult to set on scientific computing center.

[1] `http://gridtlse.org/`
[2] `http://pastix.gforge.inria.fr`

4 Presentation of DIET

DIET is a GridRPC middleware relying on the client/agent/server paradigm. A **client** is an application which needs a computing service. The **agent**, which can be extended as a hierarchy of agents, has the knowledge of several servers. The distributed scheduler embedded in each agent chooses the best computing resource for the execution of a given request considering a given metric. The **server** is a daemon running on the computing resource. The server gives performance estimations to its agent and launches a service each time requested by the client.

The mechanism to execute a request is shown in Figure 5: when an agent is contacted by a client who wants to solve a problem (a), the request follows down the hierarchy of agents to servers (b and c). They answer back performance information (c and b) which is used up in the hierarchy to determine which one suits the best the resolution of the service (b). The identity of the server is given to the client, who contacts the server and sends its data (f). Once the computation is finished, results are transfered back to the client.

5 Architecture of the proposed solution

The architecture of the solution is schemed in Figure 5. There are four main parts. In the reverse order of the process of a request: a script parameterized with correct values concerning the number of processors used in regard to both PASTIX and to the batch scheduler has to be created. This means that the server can access the number of available idle resources on the site through the batch scheduler (d); integrate a correct knowledge to decide how many of them the PASTIX service will use (d); create and transparently submit the batch script ((d) and (e)); higher in the hierarchy, the request is processed by the hierarchical scheduler, which is specifically designed for the PASTIX service (b). All this steps are described in this section.

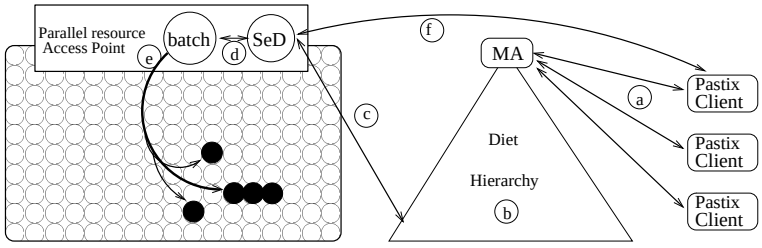

Fig. 1. Architecture of the proposed solution

5.1 Improvements realized in the DIET Grid middleware

Re-designing the implementation of DIET servers.

A server is now either a `SERIAL`, a `PARALLEL` or a `BATCH` server. A `SERIAL` server is the previous and unique kind of server available in DIET. It launches the resolution of a service by forking the function which realizes the solve. It is still the default behavior if none are given in the server code. A `PARALLEL` server reads in a file the name of the resources that it manages, and knows consequently their number. It launches a script provided by the SED programmer (generally a MPI script) where the resources and their number can be used with the help of meta-variables. It implied additional developments because the script is forked but its execution has still to be controlled for asynchronous mode, to advertise the client when the job is terminated, or simply to know when transferring back the results. A `BATCH` server reads the name of the batch system in the server configuration file, and can submit consequently adaptable batch scripts provided by the SED programmer. In that case, the SED requests the batch system every 30 seconds to control the state of the job. For the moment, OARv1.6 and Loadlever batch systems are supported.

A service is now declared and registered as *sequential* or *parallel*. This has three main purposes: 1) A `PARALLEL` or `BATCH` server can provide two different implementations, one sequential and the other parallel, with the same name. Hence, the client can request explicitly for one kind of service to explore some speed-up study, or the DIET server can dynamically choose which one to execute depending on system performances. The client does not even now the nature of the resolution nor the machine that has performed the resolution but only gets the results when the job is finished. 2) The service is registered in all the components from the server to the root agent (the Master Agent) in the hierarchy. By default, if no constraint is specified by the user, DIET tries to answer to the request with the best server (according to some metrics) specifying which type of service to execute. But if the client specifies the type of the service he wants, then the request is only forwarded down in the hierarchy to components that are aware of the service, making the scheduling process lighter and the latency smaller. 3) When arriving at the last local agent, if there is no constraint on the request and if the server can solve both kind of services, the request is duplicated when submitted to the server. Hence, one can achieve Round-Robin between services on all the platform as a default scheduling policy.

An extended API to address batch systems performance estimations.

DIET has a performance module called CoRI (Collector of Resource Information). CoRI proposes a generic API which lets the user get some given information on the system by transparently transferring the request to a software or a built-in tool. It is also used to set information that are transferred back to the agent, for application specific scheduling.

We have integrated a submodule which can get information through batch schedulers. For now, only the necessary information for the work of this paper can be ob-

tained, namely the number of processors and the number of idle processor available in the system.

A new API to interface DIET and batch schedulers.

We have extended the DIET API in order to provide means to submit a draft script constructed by the SED programmer that DIET completes and submits to the system (if PARALLEL or BATCH). Hence, DIET transparently manages batch and parallel submissions. DIET also proposes a set of meta-variables that the SED programmer can use in the draft script. They describe the system and are replaced by the SED at launch time, The script is then submitted with the additional corresponding batch syntax. Hence, the submission is conducted transparently for the client/server programmer. This gives a higher level and much ease to the server programmer to design a single generic server working on different batch systems.

5.2 Interfacing PASTIX and DIET

Interfacing PASTIX and DIET occurs at three very different levels: first, we need to use DIET functionalities to question the batch system about its load. We need as well a PASTIX performance prediction to decide how many threads per reserved processors will be used, in order to fully benefit from the moldability of PASTIX jobs; second, a DIET client/server has to be designed with the DIET batch API, in order to propose the PASTIX service to Grid users; third, due to the nature of the analysis, we focus on cycle stealing. Then the scheduling metric is based on idle resources and an application specific plug-in scheduler has to be defined.

Performance predictions.

Since the ordering step and the block symbolic factorization can be pre-calculated for each problem, PASTIX is able to quickly estimate the execution time in function of the number of processors (threads) before factorizing and solving the system. We can notice that the size of the block symbolic matrix, used to compute the prediction, is small. Moreover, at the same time, the exact memory requirement for each processor can be computed. The script can then be built by DIET to fit the best PASTIX requirements and accordingly to system performances (*e.g.*, number of processors, walltime).

Designing the DIET client/server to provide the PASTIX service:.

We have used the new DIET batch API to write the PASTIX client/server, in order to be able to submit transparently to clusters managed with OAR and to an AIX parallel machine managed with Loadleveler. Nonetheless, writing the DIET client/server is very similar to the work in [1], except that meta-variables are used in a draft script and it is launched with a special call, which completes the batch script and launches the script in place of both the client and server programmer to the batch scheduler. The draft script built by the DIET server takes into account the values given by the performance prediction to reserve the correct number of processors (threads) for the correct duration.

Scheduling PASTIX requests in DIET.

Because we plan an extensive analysis, we need the maximum available resources. In this paper, we consider a scheduling based on the availability of resources. The work is performed using the new CoRI batch submodule to get the corresponding information on the system and the plug-in scheduler functionality of DIET [2]. At each step in the hierarchy, the aggregation method sorts the servers by the maximum available idle resources.

6 Conclusion

Concerning PASTIX, all the results and data will be collected in order to select the parameters of the solver that best fit the simulations of MHD instabilities. Some improvements are currently developed into the solver to take care of NUMA (Non-Uniform Memory Access) effects. This will induce some new constraints regarding the mapping of resources during batch reservations on such architectures.

Concerning DIET, we will extend the number of recognized batch with OpenPBS and SGE batch reservation systems. Furthermore, we will provide more functionalities regarding batch integration: an improved performance prediction for given batch systems (collecting information *and* predicting system behaviors for example with the help of Simbatch [5]), as well as better autoconfiguration tools on the server side, to automatically discover batch queues and their respective information. These will lead to a better information quality, which will be used in the DIET scheduling.

We plan to use these improvements in a future work concerning the resolution of a set of experiments where the memory has to be taken into account in the scheduling process as well as when solving the problem. Furthermore, the schema of this set of experiments can be represented as a workflow whose branches can be pruned depending on temporary results. Scheduling algorithms have then to be studied and tested.

The DIET extensions developed in this work will be integrated in the LEGO demonstrator for the evaluation of the project. Furthermore, it will be adapted when possible into the next evolutions of TLSE.

References

1. Yves Caniou, Eddy Caron, Hélène Courtois, Benjamin Depardon, and Romain Teyssier. Cosmological simulations using grid middleware. In *Fourth High-Performance Grid Computing Workshop. HPGC'07.*, Long Beach, California, USA., March 26 2007. IEEE.
2. Eddy Caron, Andréea Chis, Frédéric Desprez, and Alan Su. Design of plug-in schedulers for a gridrpc environment. *Future Generation Computer Systems*, 24(1):46–57, January 2008.
3. Eddy Caron and Frédéric Desprez. DIET: A scalable toolbox to build network enabled servers on the grid. *International Journal of High Performance Computing Applications*, 20(3):335–352, 2006.
4. O. Czarny, G. Huysmans, P. Hénon, and P. Ramet. Improvement of existing solvers for the simulation of mhd instabilities. In *Numerical flow models for controlled fusion, Porquerolles, France*, April 2007.

5. Jean-Sébastien Gay and Yves Caniou. Étude de la précision de Simbatch, une API pour la simulation de systèmes batch. *RSTI - Techniques et Sciences Informatiques*, 27:373–394, 2008.
6. P. Hénon, P. Ramet, and J. Roman. On using an hybrid mpi-thread programming for the implementation of a parallel sparse direct solver on a network of smp nodes. In *Proceedings of Sixth International Conference on Parallel Processing and Applied Mathematics, Workshop HPC Linear Algebra, Poznan, Pologne, LNCS 3911*, pages 1050–1057, September 2005.
7. P. Hénon, P. Ramet, and J. Roman. On finding approximate supernodes for an efficient ilu(k) factorization. *accepted to Parallel Computing*, 2007.
8. F. Pellegrini. SCOTCH 4.0 User's guide. Technical report, INRIA Futurs, April 2005.