Vigne: Executing Easily and Efficiently a Wide Range of Distributed Applications in Grids

Emmanuel Jeanvoine[1,3], Christine Morin[2,3], and Daniel Leprince[1]

[1] EDF R&D
[2] INRIA
[3] IRISA – PARIS project-team

Abstract. Using grid resources to execute scientific applications requiring a large amount of computing power is attractive but not easy from the user's point of view. Vigne is a Grid system designed to provide users with a simplified view of a grid. This paper presents a set of system services that allow to run a wide range of distributed applications in a simple and efficient manner. A running prototype has been implemented as a proof of concept and experiments on the Grid'5000 testbed show the efficiency of our approach.

1 Introduction

Numerical simulation has an important place in several scientific fields where experimentation is impossible. Grids have become attractive for the execution of numerical simulations since they provide a large computational power. However, using a grid where resources are spread all around the world and have a dynamic behavior is still a challenge.

Several systems are designed to execute applications on a grid and in particular the distributed ones. However they suffer from several issues. Sometimes, they are dedicated to some kinds of applications. They often require some modifications in the code of the application, some re-linking with external libraries or even a specific implementation. Finally, few systems are able to deal with particular application constraints like the need for an efficient network between the application tasks.

We propose Vigne, a system to simplify the use of a grid from the users' and the administrators' point of view. We have designed this system in order to deal with the dynamic behavior of resources in a large scale environment. It is self-healing and self-organizing [1]. Thanks to an architecture built on peer-to-peer overlay networks [2], Vigne can handle several simultaneous failures and does not suffer from any single point of failure. Vigne is composed of a set of services that provide facilities for resource discovery and allocation, memory sharing, file transfers, resource interfaces and application management. With these services, Vigne supports a wide range of applications, even the legacy ones, and does not require their modification. Vigne takes into account the specific constraints of the distributed applications and is able to perform synchronized and spatial resource allocations by providing a simple application description formalism. Finally, Vigne is designed to harness all the grid models like desktop grids, peer-to-peer computing or federated clusters.

Experiments conducted on the Grid'5000 testbed [3] with a large number of nodes validate our approach and show its efficiency.

The rest of the paper is as follows. In Section 2 we present the background with some distributed application models and related works; we also introduce our approach. In Section 3, we present the Vigne features related to the execution of distributed applications on a grid. In Section 4 we present experiments of Vigne on the Grid'5000 testbed. Finally, we conclude and present future works in Section 5.

2 Background

Several kinds of applications can take advantage of the computational power of a grid. In particular, the distributed applications can use several nodes of the grid to reduce the computation time. The applications composed of a single task are not taken into account in this paper since executing them is not so challenging.

We present in this section several models of distributed applications and some related works on the execution of distributed applications in grids.

2.1 Distributed Application Models

Several paradigms can be used to develop distributed applications according to the nature of the parallelism that can be introduced in those applications. First of all, the simplest model of distributed application is the "bag of tasks". In this model, a lot of independent tasks are executed and the goal is to minimize the makespan. Neither synchronization, nor close resource allocation is required between tasks.

Master-worker applications are also composed of a large set of independent tasks. With this model, the master assigns the tasks to the workers and collects the results. The resources required for the execution must be allocated at the same time, which represents a synchronization dependency between the tasks.

Some applications, inherited from the cluster computing world, are composed of communicating tasks. Such applications are usually based on message passing environments like MPI and are used for instance to solve linear algebra computations. Other applications, designed to solve problems related to multiphysic domains, use code coupling methods. These two kinds of applications may communicate intensively and require an efficient network connection. With this model, the resources required for the execution must be allocated at the same time and must be close, from the network point of view, in order to provide high performance network communications. This represents synchronization and spatial dependencies between the tasks. Other applications have a more complex architecture and are composed of sub-applications. The sub-applications can have several architectures like: sequential, parallel, code-coupling or master-worker.

	Synchronization	Spatial	Precedence
Bag of tasks			
Master-Worker	X		
Parallel/Code-coupling	X	X	
Workflow	X	X	X

Table 1. Dependencies between tasks according to the application models

The particularity of these complex applications is that data may be exchanged between the sub-applications. These applications are often represented with a workflow. With this complex model, synchronization and spatial dependencies may be required according to the sub-application models. Furthermore, a precedence dependency is added because some sub-applications may require the output of another one to be executed.

So, it is possible to identify three kinds of dependencies between the tasks of a distributed applications that are: spatial, synchronization and precedence dependencies. This is summarized in the table 1.

2.2 Running Distributed Applications in Grids

Several grid middlewares support the execution of distributed applications since they rely on a batch scheduler. These middlewares are not presented here and we only focus on those that natively support the execution of distributed applications and we explain how they address the issues of dependencies presented before.

Globus [4] is a toolkit that provides a set of low-level features to federate resources on a grid. It allows to run distributed applications and provides a partial solution for the synchronization dependency without requiring the modification of the applications. The problem is that Globus does not provide facilities to synchronize the allocation of several resources for a parallel or a code-coupling application. Zorilla [5] is a system based on an overlay network and provides locality-aware co-allocation with a mechanism called *flooding scheduling*. However, due to the nature of the scheduling, a lot of resources may not be used and the resulting allocation might be inefficient.

Some middlewares offer attractive features to handle synchronization and/or spatial dependencies between the tasks but they impose the modification of the applications. [6] presents an architecture for co-allocation in Globus. Primitives must be added in the applications to perform an initial barrier to synchronize all the resources used before an execution. No spatial dependency is taken into account in this system. Legion [7] provides similar features but also implies the modification of the applications in order to give them access to the Legion object-oriented programming model. Vishwa [8] provides a framework to execute distributed applications with synchronization dependencies by using the concept of Distributed Pipes that requires the modification of the applications. KOALA [9] is a meta-scheduler built on top of Globus. It provides features for spatial dependencies by placing tasks close to specified data like a file server for instance. XtremWeb [10] is a platform for global peer-to-peer computing. It targets master-worker and workflow applications and provides facilities to create distributed applications with JavaRMI or XML-RPC.

Several systems handle precedence dependencies between the application tasks. These systems are mainly workflow engines like Triana [11].

3 Vigne, a Grid System Designed to Simplify the Execution of Distributed Applications

Our contribution is the design and the implementation of the Vigne system that allows to execute distributed applications with the possibility to define synchronization, spatial and precedence dependencies between the tasks and without modifying legacy applications.

In this section, we present the Vigne features to simplify the execution of distributed applications. In particular we present the Vigne features to handle the three dependencies between the tasks mentioned in Section 2.1 and the support for legacy applications.

3.1 Full Application Example

Figure 1 presents an example of application that is used in the following. The application is composed of 7 tasks. Task 1 produces a result that is used by two task groups ({2,3}and {4,5}). The result of the group {2,3}is used by 7. The result of the group {4,5}is used by 6. Finally the result of 6 is used by 7. We can notice that the two groups represent two parallel applications. Figure 2 shows the application description in an XML-like syntax that can be used to submit the example application to the Vigne system. In order to save space, the things related to the type of resources required for the execution are not included in this description.

3.2 Handling Synchronization Dependencies

Vigne allows to define several tasks in an application like one can see the 7 tasks in the example. If no constraint is given to the system, the tasks are executed independently. This means that the system performs a resource discovery and a resource allocation for each task and then it executes those tasks on their respective allocated resources. In the application example, there is no synchronization constraints between the tasks 1, 6 and 7.

If a synchronization constraint is given between a subset of tasks, Vigne performs a resource discovery for each task and a synchronized resource allocation for the tasks impacted by the constraint. When a resource is discovered for an application, Vigne ensures that this resource will not be used for another application at the same time by temporary locking it. At the end of the allocation, the unused resources are unlocked. As a consequence, the resources synchronously allocated with Vigne are available and free.

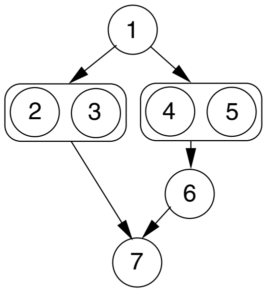


Fig. 1. Architecture of the application example

```
<application spatial_synchro="{2,3}|{4,5}">
 <task n=1>
  <task_commandline>sh exec1.sh</task_commandline>
  <task_input>input_file.dat</task_input>
 </task>
 <task n=2>
  <task_deps>1:output1.dat</task_deps>
  <task_commandline>sh exec2.sh</task_commandline>
 </task>
 <task n=3>
  <task_commandline>#</task_commandline>
 </task>
 <task n=4>
  <task_deps>1:output1.dat</task_deps>
  <task_commandline>sh exec4.sh</task_commandline>
 </task>
 <task n=5>
  <task_commandline>#</task_commandline>
 </task>
 <task n=6>
  <task_deps>4:output4.dat</task_deps>
  <task_commandline>sh exec6.sh</task_commandline>
 </task>
 <task n=7>
  <task_commandline>sh exec7.sh</task_commandline>
  <task_deps>2:output2.dat|6:output6.dat</task_deps>
  <task_output>output_file.dat</task_output>
 </task>
</application>
```

Fig. 2. Vigne description for the example application

In our example, the groups {2,3}and {4,5}have a synchronization constraint. With Vigne, we describe that with the property `basic_synchro`. Then the value of the property defines which tasks are impacted by the synchronization constraint. In the Vigne formalism, tasks that have a dependency are defined in a set and several sets can be defined by separating them with a pipe.

3.3 Handling Spatial Dependencies

To handle spatial dependencies between tasks, Vigne uses the same mechanisms that the ones used to handle synchronization dependencies but in addition, the resource allocator minimizes the distance, from the network point of view, between the resources synchronously allocated to the tasks.

Several methods can be used to measure the distance between resources in a network. We present here two methods that can be used in the Vigne system. First, it is possible to create the network matrix of the latencies between the discovered resources of all the tasks concerned by a spatial constraint. Even if that kind of method provides good results because it is really accurate, constructing the network matrix induces a significant network traffic, in particular if the grid nodes have a dynamic behavior. Indeed, in this case the network matrix cannot be created once and for good. That is why we choose to use a more simple method that uses the DNS name of the resources. We assume that the resources that share a bigger DNS suffix have a high probability to be close. For instance, the node `node22.rennes.grid5000.fr` should be closer to the node `node51.rennes.grid5000.fr` than to the node `node13.lyon.grid5000.fr`. Then, we use a backtracking algorithm that performs successive tries to find a good combination of the resources.

To specify spatial dependencies in Vigne, one must use the property `spatial_synchro` of the tag `application` and the same pattern that the one defined in Section 3.2. In Vigne, a spatial constraint induces a synchronization constraint. In our example, the groups {2,3}and {4,5}have a spatial constraint because they represent two parallel applications with intensive network communications.

3.4 Handling Precedence Dependencies

In order to run workflow applications or specific application models, Vigne provides users with features to formulate precedence relations between tasks.

When an application is submitted to Vigne, each task of the application is analyzed. If some tasks do not have precedence dependency with other tasks, Vigne performs a resource discovery for them. This is the case in our example for the task 1. For the other tasks, Vigne waits for the dependencies between these tasks to be satisfied. As a consequence, when an task completes its execution, Vigne checks if other tasks depends on this task and notifies the given tasks. When a task receives a notification, Vigne checks if all the dependencies are satisfied and if it is the case, it launches the resource discovery for this task. In our example, the tasks 2, 3, 4, 5 are executed only at the end of the task 1. The task 6 is executed at the end of the tasks 4 and 5. Finally, the task 7 is executed at the end of the tasks 2, 3 and 7. As far as resources are allocated only when a subset of the tasks is ready to be executed, no resource is wasted. However, this approach has a major drawback. If an application is composed of some tasks that require resources that are not in the grid, the execution application would be interrupted and some tasks would have been previously executed uselessly. A solution would be to reserve the resources at the submission time, but in this case, a lot of resources may be wasted. If we consider a large grid, the probability that tasks require resources that are not in the grid is quite low. So we prefer the first solution, which is best to avoid resource wasting.

When precedence dependencies between tasks are formulated, Vigne also offers the facility to transfer directly some files between the resources of the dependent tasks. This operation is possibly performed at the end of the execution of a task.

In this paper, we only focus on precedence dependencies induced by file exchanges between tasks, i.e. we deal with the case where one task produces some files that are used as an input for other tasks. Another application model where the tasks use shared memory to exchange data has been studied in [1].

To specify precedence dependencies in Vigne, one must use the tag `task_deps` of the concerned tasks. The value of the tag is as follows (in a regular expression syntax):

```
task_number:[file][,file]*[|task_number:file[,file]*]*
```

In our example, one can see that the task 7 needs the output files `output2.dat` of the task 2 and `output6.dat` of the task 6 to be executed.

We can notice that the description of our application example does not specify precedence dependencies between the tasks 1 and 3 because the tasks 2 and 3 are synchronized with a spatial, and implicitly with a synchronization, dependency. As a consequence, the task 3 will not be executed before the end of the task 1. The same occurs between the tasks 1 and 5.

3.5 Legacy Application Support

Some applications need to know the location of all their tasks when they are running. For instance, one must provide a *machine-file* to run an MPI application. As far as the location of the resources used by the application is not known at the submission time, this kind of information cannot be given by users.

Vigne provides several environment variables in the execution environment of all the resources used by the application.

These variables are `$VIGNE_MACHINEFILE` and `$VIGNE_TASK_n` (where `n` is the number of the task). `$VIGNE_MACHINEFILE` contains the path to a file where the location of all the resources used by the application are written. `$VIGNE_TASK_n` contains the DNS name of the resource allocated to the task `n`. As a consequence, the application does not need to be modified to be executed.

For instance, the script `exec2.sh` used by the task 2 in our example (see Fig. 2) may contain something like:

```
echo $VIGNE_TASK_2 > machines && echo $VIGNE_TASK_3 >> machines
lamboot -d machines && mpirun -np 2 mpi_program && lamhalt
```

The first line of the script is used to create the machine-file with the resources used by the tasks 2 and 3. The second line is used to launch `mpi_program` on two resources using the LAM MPI environment.

4 Experiments

4.1 Vigne Prototype

A working prototype of Vigne has been implemented with all the features presented in Section 3. It runs on Linux based systems and it does not rely on any middleware. Vigne is written in C and the prototype is materialized by a daemon that must be run on each grid node and a client used by grid users to perform queries like: application submission, information about execution, execution cancellation, manual file transfers between tasks, etc.

To submit an application, a user must provide an XML description like it is shown in Figure 2. To simplify the submission of applications, a GUI could be developed to generate easily the XML description.

4.2 Evaluation of the Co-Allocation mechanisms

In these experiments, we evaluate the co-allocation mechanisms of Vigne. To do that, we have used a parallel MPI application computing the number π. Implicitly, an MPI application has synchronization dependencies between its tasks since each task must be run at the same time. Furthermore, this application performs intensive network communications, so it has spatial dependencies between its tasks.

The precedence dependency is not evaluated here since it can only be evaluated qualitatively and the real difficulties rely in the management of the two other dependencies.

We perform two types of experiments. In the first one, we do not use the spatial synchronization, so we use the `basic_synchro` property in the application description. In the second one, we use the `spatial_synchro` property in order to minimize the distance between the resource allocated to the tasks. For each experiment, we launch 50 instances of the application (the instances are launched every two seconds). Each experiment is launched three times, with a different value for the number of nodes used by the MPI application. The values are 4, 8 and 16 nodes.

4.3 Setup

To evaluate the Vigne prototype, we have used the Grid'5000 [3] testbed with a large number of nodes. Grid'5000 is a French experimental testbed for research in Grid computing composed of several hundreds of nodes spread over 9 sites in France. We have used 376 nodes spread over 6 sites: Lille (31), Lyon (45), Nancy (42), Orsay (93), Rennes (115), Sophia (50). Vigne has been deployed over all these nodes.

4.4 Results

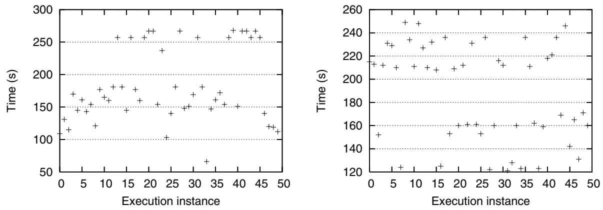
	Without spatial optimization			With spatial optimization		
	4 nodes	8 nodes	16 nodes	4 nodes	8 nodes	16 nodes
1 site	1	0	0	47	45	33
2 sites	14	0	0	3	5	12
3 sites	22	9	0	0	0	2
4 sites	13	24	1	0	0	3
5 sites	0	14	19	0	0	0
6 sites	0	3	30	0	0	0

Table 2. Occurrences of the number of sites used in a computation according to the number of nodes per application

On Figure 3, one can see the execution time of each instance of the MPI application for the experiment where the spatial optimizations have not been used.

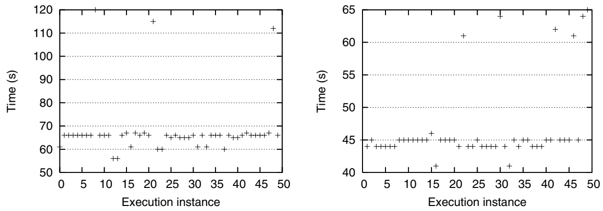
On Figure 4, one can see the execution time of each instance of the application for the experiment where the spatial optimizations have been used.

If we observe Figures 3 and 4, we can see that the spatial optimization seriously reduces the execution time of the tasks. We can also observe that an average speedup of 40% is obtained with the spatial optimizations when we use 8 nodes instead of 4 nodes. Without spatial optimization, there is no real speedup on average. With 16 nodes, the spatial optimizations do not provide as good results as with 4 or 8 nodes because the resources are much more overloaded and in this case it is hard to find 16 resources on the same site. However, the results with the spatial optimization are better than those without optimization. To confirm the results shown on Figures 3 and 4, the Table 2 shows the dispersion of the executions according to the number of sites used each time. Here, we can observe that with the spatial optimization, the great majority of the executions occurs with a resource allocation on a single site per application. Without optimization, the resources were mainly allocated on 2 sites and more. The results show that the co-allocation features of Vigne are working and are really efficient. The application used in the experiments performs intensive network communication, but this represents the worst case for a co-allocation mechanism. Indeed, if the resources used by the tasks are randomly allocated in the grid, large latencies between the nodes induce to much overhead for the overall performance, as shown in Figure 3.



4 MPI processes 8 MPI processes 16 MPI processes

Fig. 3. Execution time of the application without spatial optimizations



4 MPI processes 8 MPI processes 16 MPI processes

Fig. 4. Execution time of the application with spatial optimizations

5 Conclusion

This paper presents Vigne, a Grid system to execute simply and efficiently distributed applications in large scale grids. In particular, we propose a set of system features implemented in the system Vigne that allow to execute distributed applications without modification of the code and in an efficient way thanks to spatial optimizations in the resource allocation service. The Vigne system and the proposed mechanisms have been evaluated on the Grid'5000 testbed with a large number of nodes. The results show the efficiency of our approach.

Future works will be dedicated to the ability to execute distributed applications on a grid where resources may be located behind firewalls or private networks. Vigne will also be evaluated with industrial applications, in particular in the framework of the SALOME numerical platform.

References

1. Rilling, L.: Vigne: Towards a Self-Healing Grid Operating System. In: Proceedings of Euro-Par 2006. Volume 4128 of Lecture Notes in Computer Science., Dresden, Germany, Springer (August 2006) 437–447
2. Jeanvoine, E., Rilling, L., Morin, C., Leprince, D.: Using overlay networks to build operating system services for large scale grids. In: Proceedings of the 5th International Symposium on Parallel and Distributed Computing (ISPDC 2006), Timisoara, Romania (July 2006)
3. : Grid'5000 website. Web Page : http://www.grid5000.fr
4. Foster, I., Kesselman, C.: Globus: A metacomputing infrastructure toolkit. International Journal of Supercomputer Applications **11**(2) (1997) 115–128
5. Drost, N., van NieuwPoort, R.V., Bal, H.E.: Simple locality-aware co-allocation in peer-to-peer supercomputing. In: Proceedings of the Sixth International Workshop on Global and Peer-2-Peer Computing (GP2P). Volume 2., Singapore (May 2006) 14
6. Czajkowski, K., Foster, I., Kesselman, C.: Resource co-allocation in computational grids. In: Proceedings of High Performance Distributed Computing (HPDC-8 '99), IEEE Computer Society (1999) 219–228
7. Chapin, S.J., Katramatos, D., Karpovich, J.F., Grimshaw, A.S.: The legion resource management system. In: Proceedings of the Job Scheduling Strategies for Parallel Processing, London, UK (1999) 162–178
8. Venkateswara Reddy M., Vijay Srinivas A., G.T., D., J.: Vishwa: A reconfigurable peer-to-peer middleware for grid computations. In: Proceedings of the 35th International Conference on Parallel Processing, Ohio, USA, IEEE Computer Society (August 2006) 381–390
9. Mohamed, H., Epema, D.: The design and implementation of the koala co-allocating grid scheduler. In: European Grid Conference. Volume 3470 of Lecture Notes in Computer Science., Springer-Verlag (2005) 640–650
10. Cappello, F., Djilali, S., Fedak, G., Herault, T., Magniette, F., Nri, V., Lodygensky, O.: Computing on large-scale distributed systems: Xtremweb architecture, programming models, security, tests and convergence with grid. Future Generation Computer Systems **21** (2005) 417–437
11. Taylor, I., Shields, M., Wang, I., Harrison, A.: Visual grid workflow in triana. Journal of Grid Computing **3**(3-4) (September 2005) 153–169