A Peer to Peer Computing Framework: Design and Performance Evaluation of YML

Olivier Delannoy* and Serge Petiton‡

* *PRiSM, Université de Versailles Saint-Quentin, France*
olivier.delannoy@prism.uvsq.fr

‡ *INRIA / LIFL Université de Lille, France*
petiton@lifl.fr

***Abstract*— Peer to peer and grid systems provide attractive middlewares to solve large numerical problems. The development, deployment and execution of applications using those middlewares suffer from the lack of well-adapted advanced tools. There is not any available solution to use the same application on two distinct middlewares. Our article presents the YML Framework which provides supporting tools to design and execute portable parallel applications over large scale peer to peer and grid middlewares. The YML Framework defines a new parallel programming language called YvetteML which is composed of a graph language and a component model. We evaluate the performance of our framework with a simple numerical application using XtremWeb as a middleware.**

***Index Terms*— Parallel and Distributed Computing, Peer to Peer, Matrix Computation**

I. Introduction

Peer to peer and grid systems are middlewares that provide a set of services to aggregate an important number of resources inside a unique system. Hence it implies the necessity of managing heterogeneity, scalability, dynamics and fault recovery. Projects like *seti@home*[1] and *distributed.net*[2] have demonstrated the validity of solving some numerical applications on peer to peer systems. The Auger project [1] uses XtremWeb to simulate the highest energy cosmic ray showers. The three projects named above are based on peer to peer systems and they all share a property. Each parallel task does not exchange any data during the computation.

No reliable tools are currently available to help the user during the development of parallel applications on a large scale architecture. As a result the user has to solve by himself problems such as heterogeneity, fault tolerance, resource management and scheduling for each new application. The dynamics of such architectures increase the global difficulty of using those platforms. Those difficulties specific to each middleware should not have to be solved for each application. Advanced tools should solve most of the complexity automatically and transparently for the user by introducing more and more services in the execution layer.

We designed the YML Framework so as to provide some tools that will help users design and execute complex parallel applications. Its goals are to help users during the develoopmeint and execution steps. Through our work we aim at demonstrating the feasibility of using peer to peer and grid systems for parallel communicating applications. In addition to this we design easy-to-use tools in order to manage underlying middlewares. This article focuses on performance evaluation with a numerical application. The performance evaluation consists in two series of experimentations. The first one corresponds to the hypothesis that the application data are previously generated by a peer to peer application or available on a distributed persistent storage. The second series of experimentations integrates the peer to peer application generating the matrix involved in the computation.

The second section presents the solution in use on the different grid and peer to peer middlewares through user's point of. It mainly deals with tools and libraries available to design parallel applications. The third section introduces the YML Framework. It also shows how it interacts with grid and peer to peer middlewares and presents the first implementation using the XtremWeb peer to peer system. The fourth section relates to performance evaluation. The last section discusses on the benefits of this framework and presents future works regarding this project and the algorithmic well-adapted to such middlewares.

II. Related Work

Parallel architectures mainly provide two ways of designing parallel applications, namely the Message Passing Interface (MPI) library and the OpenMP library. OpenMP is devoted to systems including shared memory functionalities. Nevertheless, large scale architecture software does not currently provide any support regarding shared memory. MPI is targeted to distributed memory. The current MPI implementations available on high performance parallel computers are generally the 1.1 standard. This standard faces a limitation. Indeed there is no way to adapt the number of participants in the communication during the execution. The number of processors involved in a computation is fixed at the beginning of the execution. On grid and peer to peer systems, the amount of available resources may change quickly. Therefore, grid and peer to peer implementations of the MPI standard suffer

[1] `http://setiathome.ssl.berkeley.edu/`
[2] `http://www.distributed.net/`

from the limitation of the fixed number of participants. There are two ways of solving fault tolerance problems inherent to large scale architecture softwares. MPI implementations try to manage faults automatically. It is the case of implementations such as MPICH-V [2] or MPICH-G [3], [4]. The other approach consists in informing the application that a problem occurred and letting the application apply a recovery procedure or not. It means that the number of participants in those kinds of implementations may change during the execution. The FT-MPI [5] is an example of such implementations.

An emerging standard for parallel computing on grid and peer to peer architectures currently experimented and designed is a grid enabled Remote Procedure Call (RPC) API. A growing number of projects implement RPC mechanisms to address jobs submission on middlewares. At the moment the most used and mature ones are GridRPC [6], [7] and OmniRPC [8]. The GridRPC seems to be the future standard API common to numerous grid and peer to peer middlewares. An RPC based approach is also used in Netsolve and its implementation over grid middlewares: GrADSolve [9]. Applications using those kinds of API are mainly based on two different parts: on the one side a client control program and middleware registered services or functions on the other side. The XtremWeb middleware provides the P2P-RPC [10] implementation. The RPC paradigm addresses the problem of the fixed number of participants. However, communication between participants is not really integrated to the model. It is rather difficult in two distinct RPC calls to exchange data during the computation of both RPC.

An alternative approach used in some projects is based on coordination languages. Those languages distinguish two different aspects. On the one hand, the flow control is to be found and on the other hand there are computation blocks. These two clearly separated aspects improve the readability and the computer-based recognition of the program but this implies the knowledge of two languages: the one used to describe control flow and the one used for computation blocks. The flow control acts like a control program in Netsolve or in RPC based applications. As to the computation blocks, they are usually written using standard generalized programming languages like C/C++, JAVA or Fortran. This approach has been used in Opus [11] and TwoL [12], [13]. The coordination language maps the top algorithm structure and the computation maps the simple tasks. Coordination languages have been used to separate data parallelism from tasks parallelism [14]. The redundant works associated with communication are handled automatically. DagMan provides a runtime support for Condor [15] pool clusters. DagMan enables the user to describe a graph of tasks. It is not exactly a coordination language because it does not include the task description aspect but it offers capabilities of expressing the coordination between tasks.

Previously introduced API are ongoing works and active projects. The MPI approach is well suited to numerical problem solving due to the amount of existing applications ready to use. However, there are no dedicated tools available to develop applications on peer to peer and grid systems. It is not a problem as long as we use such systems without communication between participants. In such applications parallelism reaches its maximum because there is not any synchronization between participants. It consists of the same computation executed independently on each processor with a different set of initial parameters. It is called *Task farming*. The AppLeS Parameter Sweep Template project [16], [17] provides users with a way of executing a set of tasks on a grid middleware. That set is described using XML input defining the job requests. The software handles requests trying to optimize different criteria such as data migration, total execution time and the number of tasks executed at the same time. All numerical problems cannot be solved using the same techniques. Some applications need communication between participants.

In order to develop peer to peer applications the JXTA [18] project designed a set of protocols as basic blocks of peer to peer communities. Those blocks provide a generic peer to peer framework to design applications. JXTA also provides protocols for resources discovery, queries, communication between participants using a pipe, etc. This set of protocols and the corresponding implementation available define the underlying infrastructure for a peer to peer network. On top of those protocols, developers can create applications. The JXTA project creates a multiple applications layer for peer to peer applications and provides useful mechanisms to design new applications. Nevertheless, it does not directly provide a way to execute parallel applications. It does not solve problems such as schedule of applications, deployment or resource allocation. It is mainly a general set of protocols for peer to peer applications. JXTA wants to be as general purpose as possible.

III. YML Framework

Large scale architectures, mainly grid and peer to peer systems, aggregate heterogeneous computers in a complex set of computation resources. In such an environment, resources appear and disappear rather frequently. Users cannot handle the dynamics of the architecture manually. When they want to use peer to peer applications, they must be aware that they cannot make any hypotheses on the organization and on the availability of one or a group of resources. This implies that the application must adapt its computation itself to the available resources. Another problem occurring with such architectures lies within the heterogeneity of the hardwares and of the local softwares available on each participant.

The YML Framework aims at providing the user with a simple way to develop and execute applications on peer to peer and grid middlewares. One role of this framework is to define an abstraction for middlewares, hiding differences among them and using this abstraction to remain portable over multiple middlewares. The user can easily develop a complex parallel application which may transparently execute on multiple middlewares during one application execution. Figure 1 reflects the project's organization. Users interact with the framework using a web ASP or any client of the YML Framework. Using the ASP, one can define components or execute complex applications. A set of directories helps in the development of its application providing expert knowledge.

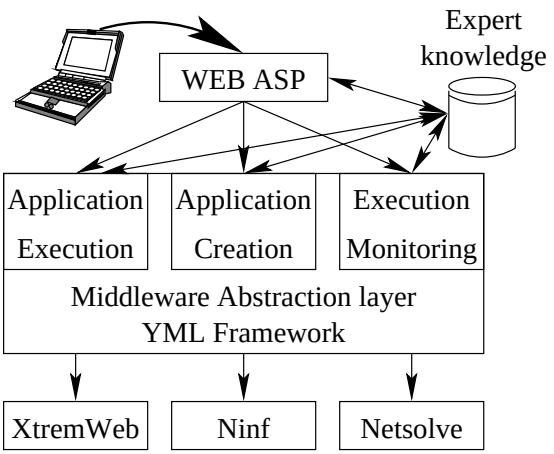

Fig. 1. Project Organisation

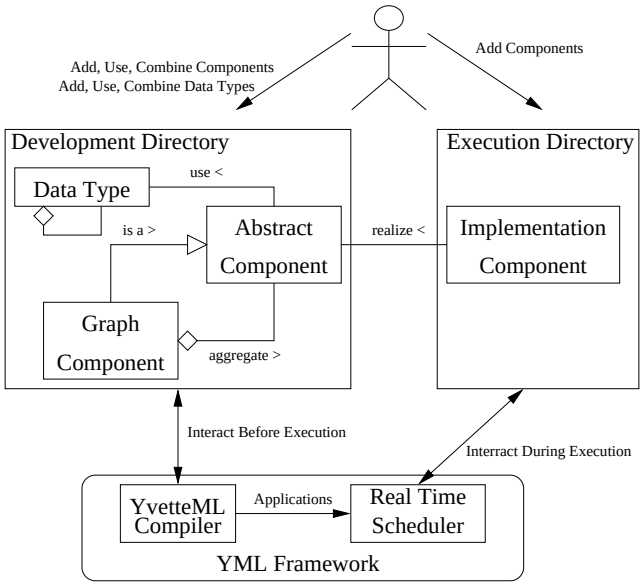

Fig. 2. Components Model and Directories

An abstraction of middlewares has to provide a set of services to the user application, which should be available on each middleware or easy to simulate on the ones which are not able to provide one of those services. We design our abstraction around a component model linked to a graph description language. The graph language is used to link together components involved in the execution of an application. The YML Framework defines a component as an encapsulation of simple tasks nodes of a directed acyclic graph representing a complex application. Components exchange data using defined input and output channels. All components are registered into directories. There are two levels of directories inside the YML software. A directory is a list of components separated into different namespaces. A global one aggregates all components available to any user. It can only be modified by privileged users. Otherwise entries in each normal user's namespace are reserved to the current user. He manages his own namespace and the information contained in it can only be accessed by himself. The namespace separation is common to both directories but the contents of these differ.

The YML Framework component model defines three different component classes as shown in figure 2. Abstract Components and Graph Components are used to provide independent application graph descriptions. Implementation Components are binary applications executed on middlewares. These are described using an automatically generated skeleton and a user provided computation code. This computation code can be written using languages such as C/C++, Fortran or JAVA depending on the availability of skeleton generators for the targeted language. The computation code does not include the communication mechanism, data encoding and other services used to manage heterogeneity of the peer to peer or grid middleware. Components are described using XML. Generators are written using XSLT stylesheet transformations.

The first directory is used for the development stage. It contains objects needed during the creation of an application and can be shared between many underlying middlewares. It also encompasses a set of blocks usable to design applications. The content of the Development Directory does not depend on the underlying middlewares. The second directory contains middleware specific information such as binary applications. Figure 2 describes the contents of each directory and the way the user can interact with both of them. The figure also introduces two modules composing the YML Framework. The compiler takes in input the YvetteML language and generates new components, applications and data types going in directories or to the real time scheduler. The scheduler controls the execution of applications. Both modules will be described further on.

During the execution stage, the execution directory is solely used. This straight separation enhances portability of the global system and makes it possible to postpone some important choices until execution time depending on the current state of the middleware. Components are separated into three categories. Those categories are introduced to solve different problems like reusability, portability and adaptability.

The graph description language provides constructions to describe a parallel application composed of previously introduced components. Its goal lies in describing dependencies between components on one side and parallelism in the application on the other side. A graph is a comprehensive representation of a template for a parallel algorithm. The template is filled using a set of components corresponding to a specific application.

To conclude the YML Framework overview, we present an example of an YvetteML application. This example mainly presents the graph description language aspect. It does not deal with components description. Figure 3 contains the YvetteML program. The XML document begins with the root element **yml** providing user accounting information. Accounting information is used to manage namespace in the YML Framework. An application always starts with an **application** element. This example hides details concerning application management such as the name used to identify results. The YvetteML language used to describe graphs is contains in the **source** element. It contains two parts. Declarations stand first. They

provide construction to define constants, events and variables. Declarations help the compiler to validate the application before its execution. The second part of an YvetteML application consists in statements. They provide constructions to call components, create parallel sections and synchronize executions of parallel sections.

```
 1 <?xml version="1.0"?>
 2 <!--
 3    top level element for
 4    YvetteML program
 5 -->
 6 <yml user="login" password="pass">
 7  <application>
 8   <source>
 9 #Decl.: const integer value
10 const problemSize := 10000;
11
12 #Decl.: event array of dimension 2
13 event evt[2];
14
15 #Decl.: variable of type MatrixReal
16 var MatrixReal vRes[1];
17
18 #Statements start after declarations
19 par # a parallel section
20  # an iterator based parallel section
21  par (i:=1, problemSize) do
22    # a component call
23    compute fillMatrixReal(vRes[i],
24      problemSize, i);
25    # notify an event to the whole
26    # application
27    signal(evt[i,1]);
28  end par do
29 // # Parallel block separator
30  # Block until the events named
31  # evt[10,1] and evt[11,1] are
32  # notified.
33  wait(evt[10,1] and evt[11,1]);
34  # a component call
35  compute SumMatrixReal(vRes[10],
36    vRes[11]);
37  #...
38 #//
39 #  ...
40 end par
41     </source>
42  </application>
43 </yml>
```

Fig. 3. YvetteML Application Template

A. Implementation

The YML Framework interacts with the user using a compiler. This one translates components into binary applications and directory entries. It also translates graph descriptions in YvetteML into fully expanded graphs ready for execution over the underlying middleware. Figure 4 provides an overview of the inside of the YML Framework. The two main modules composing the YML Framework are the *compiler* and the *scheduler*. The compiler ensures portability while the scheduler provides execution management. It analyses graphs from the *compiler* and manages dependencies for the middleware. The *scheduler* interacts with the middleware using both tasks *launcher* and *retriever* modules. The YML Framework acts as a normal client for the execution layer.

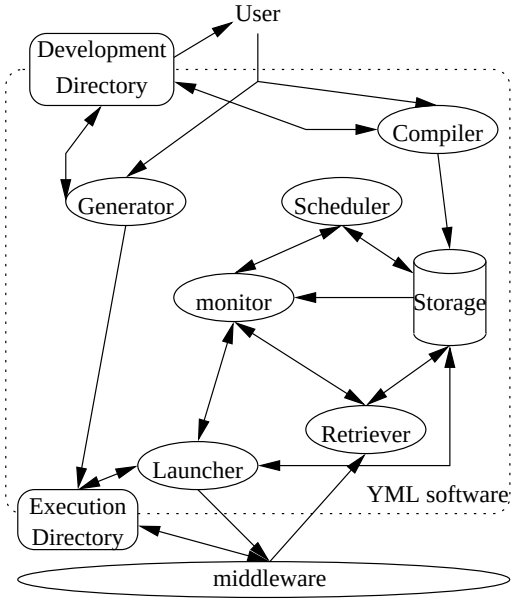

Fig. 4. YML Software Organization

The XtremWeb middleware is a peer to peer environment which provides support for binary applications execution and deployment. The YML Framework simulates communication between workers defining a centralized shared memory mechanism. This shared memory is a persistent storage used to store partial results of each job executed on the XtremWeb middleware. The shared memory can be considered as an incremental checkpoint of the computation data. This centralized implementation introduces a bottleneck. For each component execution, data come from the shared memory located on the same computer as the YML Framework and go back after updates to the YML Framework hosting nodes so that partial results should be available to dependent component execution. There is not any direct communication between peers. Obviously it is transparent for the user ; so if a middleware provides built-in support for communication between workers, the YML Framework backend dedicated to this middleware overrides the YML general functionning in order to benefit from these functionnalities. The YML Framework scheduler modules monitor the state of the XtremWeb middleware in order to accurately create job requests. Dependencies between tasks are managed using a table of events. Each YML task is decorated using an execution condition and a post execution events set. This condition is a boolean expression depending on the events involved in the computation of an application.

Once the boolean expression is true, the task is ready for execution. After the computation on the middleware, all events composing the execution events set associated with the task are notified making the application progress.

IV. Experimentations

We evaluate our software using the application $y = Ax + x$ where A is a square dense matrix composed of real numbers and x and y are both vectors of real numbers. Supposing the A matrix does not fit in the memory of a peer, we divide A in a set of blocks. Each block is itself a square matrix. We define bs as the size of one block and bc as the number of blocks on one dimension. The number of elements of A is $(bs * bc)^2$. Vectors are also divided in blocks. The size of each block is bs too. The algorithm of the application shown in figure 5 explains its steps. The first parallel section generates and duplicates the initial vector. The copy stands for the initial value of the resulting vector. After this parallel section, the resulting vector contains the initial copy of x. The second parallel section works in columns. Each column of the matrix is computed before the next column and the resulting vector is updated after each step. In figure 5 bs does not appear because it only has an effect during the execution of the block matrix vector multiplication component at line 9 of the algorithm. The YvetteML program will illustrate this point in figure 7.

```
 1  in parallel do
 2      for i from 1 to bc do
 3          y[i] = x[i]
 4      end for
 5  end do
 6  for i from 1 to bc do
 7      in parallel do
 8          for j from 1 to bc do
 9              y[i] = y[i] + A[i,j] * x[j]
10          end for
11      end do
12  end for
```

Fig. 5. Algorithm of the $y = Ax + x$ Application

Figure 6 explains the order of execution and data migration involved in the application. All arrows using the same shape and line style are executed in parallel. Numbers on top of those arrows symbolize the current steps. The figure also introduces the components involved in the application.

In those experimentations, the algorithm does not introduce the maximum degree of parallelism available in the computation of $y = Ax + x$. The most parallel version should compute all blocks of the matrix independently and then reduce each result. In this algorithm we merge the computation on each block and the reduction. We introduce those constraints in order to keep a simple application with an important number of dependencies between tasks.

A. Experimentations with XtremWeb

Experimentations evaluate the YML Framework over XtremWeb on two separated platforms. The first one aggregates a dispatcher and 6 peers connected to each other on a local network with an ethernet 100Mbit/s network. Peers have the same hardware configuration except for one. With this small platform we evaluate the YML Framework with an important number of tasks on each peer. The second platform[3] gathers one hundred and twenty-two peers. Hardwares are heterogeneous as for the ethernet network speed composed of both 10Mbit/s and 100Mbit/s links. The second platform evaluates the global performances when the number of peers is similar to the number of tasks ready for execution at a time. Both of them are in production stage. Experimentations have been done with normal use of the network and users activities on computers. Table I describes computers hardware composing both platforms.

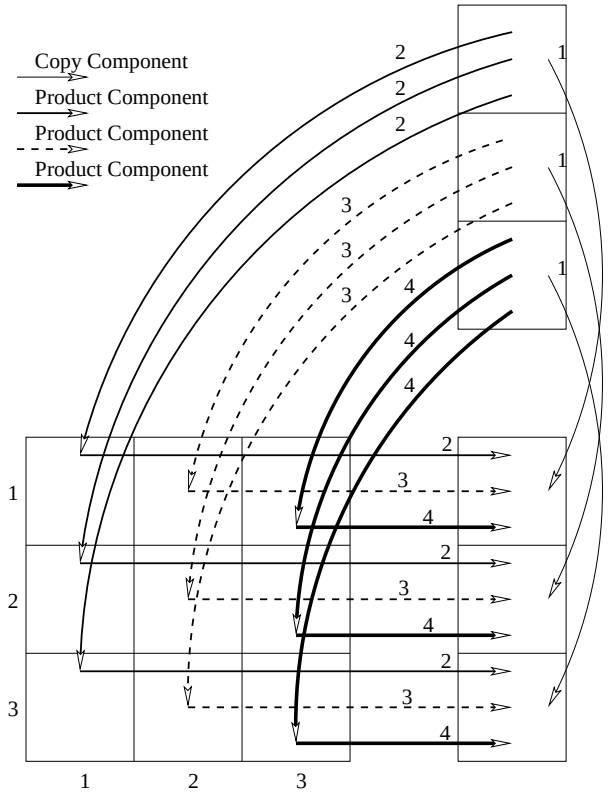

Fig. 6. Application Execution Order with bc equal to 3

The first series of experimentations is common to both platforms. It supposed that matrix A was previously located on peers. This hypothesis corresponds to the existence of peer to peer storage mechanism or of peer to peer matrix generation application. In our application the A matrix is used only once during the block based matrix vector multiplication and each block of A is also used only once. This property permits us to merge the block generation and the $y_{colN}[j] = A[j, colN]x[j] + y_{colN-1}[j]$ in the block based matrix vector multiplication component. It is a way to simulate data prefetching, matrix generator peer to peer application or peer to peer data warehouse. The amount of communication between peers consists only in both x and y vectors. Figure 7 contains the YvetteML program used for this series of

[3]All computers are located at Polytech'Lille (EUDIL)

6 peers (PRiSM)		
Count	CPU	Memory
5	Intel Pentium IV 1.8GHz	256MB
1	Intel Pentium III 800MHz	256MB
122 peers (EUDIL)		
Count	CPU	Memory
28	Intel Pentium III 450MHz	128MB
28	Intel Celeron 2.2GHz	512MB
23	Intel Celeron 2.4GHz	512MB
15	AMD Duron 750MHz	256MB
14	Intel Celeron 600MHz	128MB
8	Intel Celeron 2GHz	512MB
8	Intel Celeron 1.4GHz	256MB
4	Intel Pentium IV 2.4GHz	512MB

TABLE I
COMPUTERS HARDWARE HOSTING A WORKER FOR BOTH PLATFORMS

experimentations. It is a direct transcription of the algorithm described in figures 5 and 6. bs and bc appear respectively as *blockSize* and *blockCount*.

Table II shows results concerning these experimentations on both platforms. The first two columns describe the size of the problem expressed using bc and bs as defined previously. The third column contains the amount of data transferred over the network during the computation. The fourth column displays the number of tasks executed on the middleware. The fifth column shows the time needed to compute the application on the platform composed of six peers. The last column provides the same information as the previous one for the platform at EUDIL composed of one hundred and twenty two peers. The size of communication does not integrate XtremWeb internal needs. It corresponds to the sum of the size of data exchanged between the node hosting XtremWeb Dispatcher and the peers involved in the application computation. The information size is not measured on the network, it is the theoretical result which is a function of two parameters bc and bs.

$y = Ax + x$ **with** A **matrix generated in the matrix vector multiplication component.**					
bc	bs	**Com.**	**Tasks**	**6 P**	**122 P**
20	100	1Mb	440	6mn	6mn
50	100	4Mb	2600	18mn	17mn
20	1000	7Mb	440	6mn	6mn
50	1000	39Mb	2600	114mn	19mn
200	1000	615Mb	40400	14h	8h

TABLE II
PERFORMANCE FOR $y = Ax + x$ OF SIZE $(bc * bs)^2$

Those experimentations put forward that all the applications generate an important number of tasks and compute in less than a day's work. The amount of data handled is important due to the centralized organization of YML over XtremWeb, which for each task implies that data go to and fro between the node handling the YML Framework and workers. XtremWeb does not provide mechanisms to associate a job to a worker. It also implies that all peers do not necessarily participate in the computation of an application. Both platforms evolve in the same way except for the fourth row which corresponds to an increased day's work activity on the network and peer load.

```
 1 #Compute y = Ax + x
 2 #Declaration
 3 #Number of block
 4 const blockCount := 100;
 5 #Size of block
 6 const blockSize := 1000;
 7 # a two dimension event set
 8 event steps[2];
 9 var VectorReal vResult[1]; #computation
10 var VectorReal vInput[1];  # vars
11 #Begin of the computation
12 par
13   par (i:=1, blockCount) do
14     #create x vector
15     compute GenVectorReal(
16       vInput[i], i, blockSize);
17     # notify the block i is available.
18     signal (steps[-1, i]);
19   end par do
20 // # parallel block separator
21   par (i:=1, blockCount) do
22     # initialize y: first step of
23     # the algorithm
24     wait (steps[-1, i]);
25     # y = x
26     compute Copy(vInput[i],vResult[i]);
27     signal (steps[0,i]) ;
28   end par do
29 //
30   par (i := 1, blockCount)
31       (j := 1, blockCount) do
32     wait(steps[i - 1, j]);
33     # local y = Ax + y
34     compute GenProduct(vResult[j],
35               vInput[j], i, j);
36      signal(steps[i, j]);
37   end par do
38 end par
```

Fig. 7. $y = Ax + x$ YvetteML Program Used In First Series Of Experimentations

Figure 8 displays the table II in a graphical way. The second and third marks of both curves show how performances may vary according to the distribution of the data and the number of tasks. The loss of performance for bc equal to 50 and bs equal to 100 is due to latencies internal to the YML Framework monitor and to XtremWeb internal functionning. It points at how important matrix distribution is and therefore bc and bs are critical parameters for performances.

In the second series of experimentations we separate the matrix generation and the block based matrix vector multiplication. We add to our application a component with the role of generating the matrix. In those experimentations, each block of the matrix is generated randomly. The results would have been similar if each block of the matrix had been acquired from a data warehouse accessible using networks

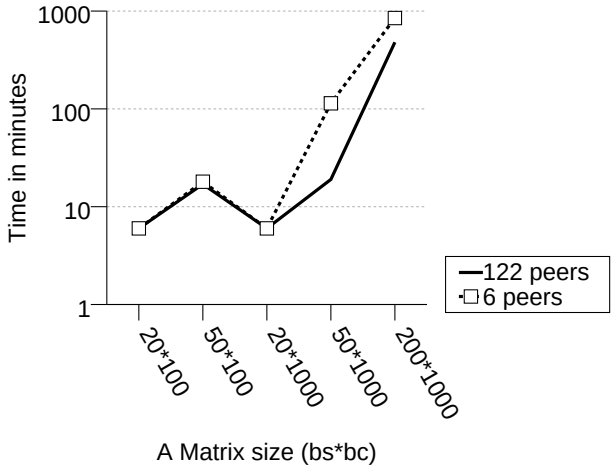

Fig. 8. Executions with Matrix previously located on peers

communication. This series of experimentations uses only the first small platform. The node hosting the XtremWeb Dispatcher in the second platform does not provide enough storage capacity for those experimentations. Table III provides results of experimentations. Columns are organized in the same way as in figure II except for the last column.

$y = Ax + x$ **with** A **matrix generated in a dedicated component.**				
bc	bs	**Com. Size**	**Tasks**	**6 peers**
20	100	62Mb	840	9mn
50	100	387Mb	5100	85mn
20	1000	6Gb	840	11h
50	1000	37Gb	5100	61h

TABLE III
PERFORMANCE EVALUATION WITH A SEPARATE MATRIX GENERATOR OF SIZE $(bc * bs)^2$

The amount of data exchanged during the computation is really important. This is partially due to the actual limitation of the YML Framework implementation. Currently, the application data space is managed on the host running the YML Framework or the XtremWeb Dispatcher. This implies the data go to and fro between peers and this special node, which is the bottleneck of the current implementation. Each block of uncompressed matrix needs around 7MB with bc equal to 1000. The amount of data transferred on the network during the generation of the matrix is really important. Integrating advanced compression mechanism is really interesting in this context. It increases the computation time on each peer on one side and on the other side it decreases the amount of data to transfert over the network. One block of the A matrix is needed for each block based matrix vector multiplication component call. The amount of data transferred puts forward how important advanced storage mechanisms in peer to peer middlewares are. Our future works are evaluating methods to dispatch jobs according to static data located on specific peers.

As expected, performances evaluation displays the limitation of the peer to peer architecture when the number of communications is really important rated to the number of calculus on each data. The application $y = Ax + x$ does not permit to overlap communication time with computation time. The number of operations is around twice the number of communications. However, the first series of experimentations shows that it is possible to compute the application on huge matrices while there is no need to migrate the matrix. Peer to peer systems are based on cycle stealing, but current evaluation models do not take into account the use of computers during their idle time. So, a new evaluation model is needed.

V. CONCLUSIONS AND PERSPECTIVES

The YML Framework provides support for parallel applications over peer to peer or grid middlewares. The user of the framework can express and execute complex applications using a component model and a graph description language. This graph corresponds to a parallel application where each node is a computation on the underlying middleware and each edge is a communication. The component model ensures a strict separation between information dedicated to a middleware and information portable over many of them. The YML Framework hides differences between execution layers so that the user does not have to take into account specificities of the middleware used to execute jobs during the development of his application. Users can also benefit from the components already available to build their own application. They do not need to take into account peer to peer or grid middleware specificities.

Experimentations show that the YML Framework generates an important number of tasks automatically on a significative number of peers. The Framework manages the execution of an application self-adapting to the amount of resources available in the middleware. We deliberately chose a difficult application. The amount of computation involved in this application is not important enough compared to the amount of communication. There are only a few operations for each data transferred. This is the main reason for the large time needed to compute not so large problems. However, while peer to peer systems are mainly based on a cycle stealing paradigm, new performance evaluation models need definitions for those systems. The main objective of the YML Framework is to provide the user with a way to create and execute communicating applications automatically. The application well demonstrates the YML Framework and puts forward current limitations of peer to peer and grid middlewares.

Future works focus on different aspects of the YML Framework. Its current organization is centralized. This introduces a bottleneck in the application data space management. We are evaluating different ways to distribute scheduling of the application directly into tasks executed on the middleware taking into account tasks neighborhood information only. We also study how to add storage capacities on computation peers and transform a task based scheduling to a data based one. In parallel to those aspects internal to the YML Framework we also evaluate the capacity of peer to peer architecture to solve linear algebra numerical problems from an algorithmic

point of view in a first stage. In a second stage we will adapt the YML Framework according to the evolution of the services available into middlewares with the objective to obtain acceptable performances for real applications.

VI. Acknowledgements

We would like to thank the French Ministry of Research for its support to this project involved in the ACI GRID. We also thank the Ecole Polytechnique de Lille for providing an experimentation platform.

References

[1] O. Lodygensky, A. Cordier, G. Fedak, V. Neri, and F. Cappello, "Augernome and xtremweb: Monte carlos computation on a global computing platform," *ECONF*, vol. C0303241, p. THAT001, 2003.

[2] G. Bosilca, A. Bouteiller, F. Cappello, S. Djilali, G. Fedak, C. Germain, T. Herault, P. Lemarinier, O. Lodygensky, F. Magniette, V. Neri, and A. Selikhov, "Mpich-v: toward a scalable fault tolerant mpi for volatile nodes," in *Proceedings of the 2002 ACM/IEEE conference on Supercomputing*. IEEE Computer Society Press, 2002, pp. 1–18.

[3] I. Foster and N. Karonis, "A grid-enabled MPI: Message passing in heterogeneous distributed computing systems," in *Proceedings of SC'98*. ACM Press, 1998. [Online]. Available: citeseer.nj.nec.com/foster98gridenabled.html

[4] I. Foster, J. Geisler, W. Gropp, N. Karonis, E. Lusk, G. Thiruvathukal, and S. Tuecke, "Wide-area implementation of the Message Passing Interface," *Parallel Computing*, vol. 24, no. 12–13, pp. 1735–1749, 1998. [Online]. Available: citeseer.nj.nec.com/foster98widearea.html

[5] G. E. Fagg, E. Gabriel, Z. Chen, T. Angskun, G. Bosilca, A. Bukovsky, and J. J. Dongarra, "Fault tolerant communication library and applications for high performance computing," in *Los Alamos Computer Science Institute Symposium (to appear)*, 2003. [Online]. Available: http://icl.cs.utk.edu/news_pub/submissions/lacsi2003-ftmpi-fagg.pdf

[6] K. Seymour, H. Nakada, S. Matsuoka, J. Dongarra, C. Lee, and H. Casanova, "GridRPC: A remote procedure call API for grid computing," in *the 3rd International Workshop on Grid Computing (Grid'02)*, 2002, pp. 274–279. [Online]. Available: citeseer.nj.nec.com/seymour02gridrpc.html

[7] S. Matsuoka, H. Nakada, M. Sato, and S. Sekiguchi, "Design issues of network enabled server systems for the grid," *Grid Computing*, 2000.

[8] M. Sato, M. Hirano, Y. Tanaka, and S. Sekiguchi, "OmniRPC: A Grid RPC facility for cluster and global computing in OpenMP," *Lecture Notes in Computer Science*, vol. 2104, pp. 130–137, 2001. [Online]. Available: citeseer.nj.nec.com/sato01omnirpc.html

[9] S. S. Vadhiyar and J. J. Dongarra, "Gradsolve - rpc for high performance computing on the grid," in *Proceedings of the 9th International Euro-Par Conference*, vol. 2790. Springer-Verlag, 2003, pp. 394 – 403.

[10] S. Djilali, "P2p-rpc: Programming scientific applications on peer-to-peer systems with remote procedure call," in *Cluster Computing and the Grid (CCGrid 2003), 3rd IEEE International Symposium on*, 2003. [Online]. Available: http://www.lri.fr/~fedak/XtremWeb/downloads/p2p-rpc.ps.tar.gz

[11] B. Chapman, M. Haines, P. Mehrotra, H. Zima, and J. V. Rosendale, "Opus: A coordination language for multidisciplinary applications," Institute for Software Technology and Parallel Systems, University of Vienna, Tech. Rep. ICASE Report 97-30, 1997. [Online]. Available: citeseer.nj.nec.com/article/chapman97opus.html

[12] T. Rauber and G. Runger, "Scheduling of multiprocessor tasks for numerical applications," in *Proceedings of the 8th IEEE Symposium on Parallel and Distributed Processing, SPDP 96*, 1996, pp. 474–481. [Online]. Available: citeseer.nj.nec.com/rauber96scheduling.html

[13] ——, "A coordination language for mixed task and data parallel programs," in *Selected Areas in Cryptography*, 1999, pp. 146–155. [Online]. Available: citeseer.nj.nec.com/rauber99coordination.html

[14] H. E. Bal and M. Haines, "Approaches for integrating task and data parallelism," *IEEE Concurrency*, vol. 6, no. 3, pp. 74–84, July-September 1998. [Online]. Available: citeseer.nj.nec.com/bal98approaches.html

[15] T. Tannenbaum, D. Wright, K. Miller, and M. Livny, "Condor – a distributed job scheduler," in *Beowulf Cluster Computing with Linux*, T. Sterling, Ed. MIT Press, October 2001.

[16] H. Casanova, G. Obertelli, F. Berman, and R. Wolski, "The AppLeS Parameter Sweep Template: User-level middleware for the grid," in *Proceedings of the Super Computing Conference*, 2000, pp. 75–76. [Online]. Available: citeseer.nj.nec.com/casanova00apples.htm

[17] H. Casanova, M. Kim, J. S. Plank, and J. Dongarra, "Adaptive scheduling for task farming with grid middleware," in *European Conference on Parallel Processing*, 1999, pp. 30–43. [Online]. Available: citeseer.nj.nec.com/casanova99adaptive.html

[18] B. Traversat, A. Arora, M. Abdelaziz, M. Duigou, C. Haywood, J.-C. Hugly, E. Pouyoul, and B. Yeager, "Project JXTA 2.0 super-peer virtual network," Project JXTA, Sun Microsystems, Inc, Tech. Rep., 2003. [Online]. Available: http://www.jxta.org/project/www/docs/JXTA2.0protocols1.pdf