GridMPC: a Service-Oriented Grid Architecture for Coupling Simulation and Control of Industrial Systems

Benoit Philipps
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, E3B5A3, Canada
benoit.philipps@unb.ca

Eric Aubanel
Faculty of Computer Science
University of New Brunswick
Fredericton, NB, E3B5A3, Canada
aubanel@unb.ca

Anna Healy
Department of Mechanical Engineering
University of New Brunswick
Fredericton, NB, E3B5A3, Canada
anna.healy@unb.ca

Andrew Gerber
Department of Mechanical Engineering
University of New Brunswick
Fredericton, NB, E3B5A3, Canada
agerber@unb.ca

ABSTRACT

It is now commonplace for industrial systems to be automatically controlled by computers. Furthermore, it is becoming more common for industrial systems to use Model Predictive Control (MPC) software to optimize the behaviour of a physical process through manipulation of control variables. It has been shown that MPC control can be improved through coupling with Computer Aided Engineering (CAE) simulations. We propose a grid architecture, GridMPC, to provide MPC coupled with CAE simulations for the real-time control of industrial systems. GridMPC bridges the gap between powerful simulation tools and actual physical processes. A job management service makes computing resources available to the grid for CAE simulations and a repository service stores the simulation results to avoid identical queries and provide immediately the data to the process controller. GridMPC has been built using web services, taking security in consideration. The overhead of the grid architecture is found to be acceptably small.

Categories and Subject Descriptors

D.2.11 [**Software Engineering**]: Software Architectures—*Domain-specific architectures*; J.6 [**Computer Applications**]: Computer-Aided Engineering—*Computer-aided manufacturing*; J.7 [**Computer Applications**]: Computers in Other Systems—*Industrial Control*

1. INTRODUCTION

With the development of microelectronics and the always increasing need for precision and efficiency in manufacturing systems, many industrial systems are now automatically controlled by computers. Control software requires knowledge of how the industrial process responds to changes in control variables. This data is typically obtained from experimental open loop tests. However, the number of experiments that can be run is generally limited by cost and time. For this reason most process controllers are valid only over a narrow range of conditions. However, by using Computer Aided Engineering (CAE) simulation tools to generate the open-loop data, a wide range of processing conditions can be explored and stored. When conditions in the real process are changed, the CAE tool is executed in order to generate new data for the controller. This results in more efficient and reliable control of the industrial system [6].

Manufacturers have started to make use of distributed computational resources, from desktop workstations to large parallel computers, to solve challenging problems that arise in design and manufacturing. We propose to integrate manufacturing control software into these corporate computational grids, to deal with the practical problems with the coupling of CAE and control software. Simulation tools for most advanced industrial processes are designed to reflect the real process to a large degree of accuracy, and hence can consume a significant amount of computing time. The computational resources required to produce useful CAE simulations are not likely to be co-located with the facility operating the controlled process. In addition, industrial systems operate in real time, and may need data each time processing conditions are changed. We propose our grid architecture to address these concerns.

Computational grids enable the seamless integration of heterogeneous and geographically distributed software resources. Scientific instruments are also being integrated into grid environments[5]. This work has focused so far on the use of remote instruments as producers of data that can be stored, analyzed, and used by software applications. The present work incorporates physical processes as consumers of computational resources.

There is an increasing body of experience supporting the

viability of using web services to build computational grids. For example, the majority of the UK's E-Science program participants have built their grid infrastructure using web services [4]. Web services are available in several mature implementations, and offer the functionality required to build grid environments. Additional web service standards have been proposed, such as WS-ResourceFramework (WSRF) and WS-Notification (WSN), which address the requirements associated with transient and stateful services. These are being implemented in the Globus Toolkit[7], but other implementations are available as well, including WSRF.net [14]. The latter implementation is relevant to the present work in that it proposes an architecture for remote job execution in grid environments. There have been performance issues raised about web services, but recent work has indicated that potential limits to performance may not be serious in practice [12].

There is an element missing in GridMPC which we are addressing in ongoing work, namely the feedback of information on the industrial process to the CAE model. The accuracy of the CAE model depends on its boundary conditions, which can be optimized using information obtained from the industrial process via a monitoring service.

In the next section we briefly review process control. We present an overview of our GridMPC architecture in Section 3. Detailed descriptions of the services are given in Section 4. Implementation details and performance tests are presented in Section 5, and Section 6 concludes.

2. PROCESS CONTROL

Model predictive control (MPC) algorithms apply a series of control moves to manipulated variables in order to achieve a particular objective[10]. For example, in a heating process the objective could be to set the temperature of an object, and the control variables could be the flux in the heaters that surround the object. These algorithms can be used to control a wide variety of processes, and can handle multivariate inputs and outputs. They have proved to be successful in a number of industrial applications. One disadvantage of MPC is that it requires a model of the process, describing how it responds to changes in the control variables. As an alternative to costly experimental tests, CAE simulations can be used to construct control models. The coupling between MPC and CAE can be either active or passive. In active coupling the CAE simulation runs concurrently with the process. Since for most industrial applications the CAE tools are designed to reflect the real process to a large degree of accuracy, the CAE tool requires long simulation times, much longer than the time scales of the process. For this reason a passive approach to coupling is appropriate, and as much physics as needed can be embedded in the CAE tool without concern for computing time. In passive coupling the CAE tool operates off-line and cycles through a wide range of process parameters, producing a series of open loop tests. The controller constantly access this stored data as process conditions change. The CAE simulations and process control cannot be completely decoupled, however, since situtations may be encoutered which were not covered in the off-line open loop tests. These considerations motivated the grid architecture, which is presented in the next section.

3. GRIDMPC ARCHITECTURE

Two fundamental services are required. One service has to interact with the computing resource (in the general case a computer cluster) and handle submission and management of tasks, allowing open loop simulations to be executed. In the following it will be called the Job Management Service. A second service has to be linked to a database, allowing insertion or retrieval of data corresponding to the results of open loop tests. This will be called the Open Loop Repository Service.

Another part of the application has to be located directly on the computer controlling the industrial process, in order to interact with the process control software, and then call the other services. The tasks involved in communicating with both services are quite complex: calling the open loop repository service to search for open loop data, decide whether a new simulation needs to be launched, wait for its results, and insert the results in the database. This complexity led us to separate these tasks in two parts, having on one side a Process Control Service that handles all the communications with the Job Management Service and the Open loop Repository Service, and on the other side a small client that only interacts with the process control software and calls the Process Control Service whenever the controller needs new open loop data. Figure 1 shows the architecture of the application and the interactions between services.

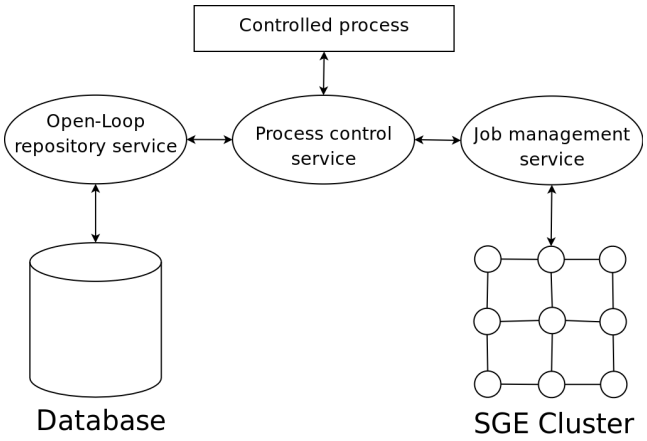

Figure 1: Grid architecture

3.1 Choice of Web Services

Having designed the general shape of the system and the composition of the tasks, we had to define which standard we would use. This choice had to be made mostly between grid services using WSRF and pure Web Services. This choice was mainly guided by the size of the architecture.

Grid environments like the Globus Toolkit[7] are aimed at building any kind of computational grid. Even though Globus Toolkit 4 is based on WSRF, it appeared that the relatively small size of our application did not require the overpowerful but complex tools Globus provide. Using WSRF.net or even making our own implementation of WSRF would have been possible, but the relative youth and the constant evolution of this framework could have made the task more

difficult[4], which added to the complexity and overhead WSRF appears to involve[14].

This led us to choose web services. We believe the overhead involved by the use of web services will be acceptable for our application and that web services provide the required functionality, such as stateful transient services, to implement our grid architecture.

3.2 Authentication and security

A key security element is that the client be authenticated by a name and a password. However, as the number of calls to services can be quite high, sending passwords repeatedly through a network was not a good idea. This is why we decided to use session tickets. Each call to any service has to be authenticated by the session ticket as first argument. The tickets have a limited lifetime and are managed by a specific service called the Single Sign-On Service that was added to the architecture. This service emits session tickets when called by the clients (here authenticated by name and password). Each other service calls the Single Sign-On service to check the ticket it received from a client.

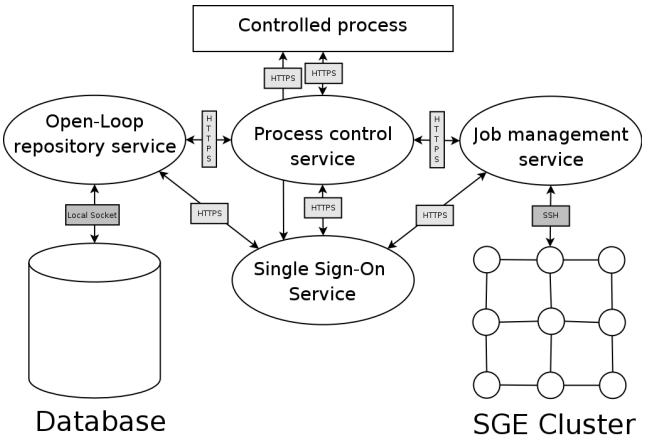

Figure 2: Integration of the single sign-on service in the grid architecture

Moreover, as a matter of trust, each service wanting to authenticate a ticket it received has to authenticate itself using a server key. Lists of server keys and users accounts are kept by the Single Sign-On service. Another element of security is transport encryption[11]. The WS-Security standard requires SOAP messages to be sent through encrypted HTTPS protocol instead of plain text HTTP. All the messages sent between two services and between a client and a service have to be sent over HTTPS. Figure 2 shows how messages are sent between services, and how the Single Sign-On service integrates into the grid architecture.

4. DESCRIPTION OF THE SERVICES

4.1 Job Management Service

The goal of this service is not only to provide job submission capability to GridMPC, but also to serve as a general job submission service, capable of submitting jobs and handling files for any application to be run on a cluster controlled by Sun's Grid Engine scheduler. Therefore we will present it in greater detail than the other services. In GridMPC, the job management service handles all the open loop simulations that need to be launched on the cluster. The cluster the service is linked to is a Sun V60 Cluster, Chorus, housed by the Advanced Computing Research Laboratory[1] at the University of New Brunswick.

The scheduling of tasks is done by Sun's Grid Engine (SGE) software [13]. This scheduler allows three main commands to be run on the shell to manage jobs: **qsub** to submit a job to the queue, **qstat** to check the status of all jobs or one user's jobs, and **qdel** to remove a job from the queue or kill it. To submit a job, a batch file containing all information about the job to be launched has to be given as an argument to the **qsub** command.

An important point is that the only access allowed to the cluster is an ssh connection to the master node, for security reasons. This means that to launch a job, a user has to log in using an ssh client, and launch the Grid Engine commands in the shell. Moreover, there is no API allowing an application to communicate directly with Grid Engine without the shell.

For our purpose, this means that the web service was not installed on the master node of Chorus itself but on another machine hosting an application server providing web services. The service itself connects to the master node of Chorus using an ssh client, and communicates with Grid Engine through shell commands, as presented in figure 3. An ssh client had to be integrated into the service and handle all communications with the cluster, using ssh to communicate with the shell, and sftp to upload and download files.

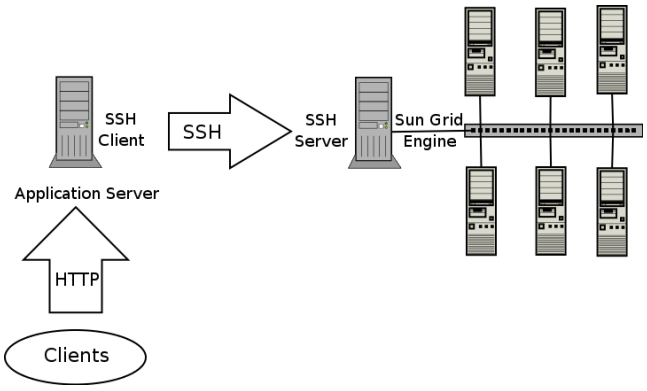

Figure 3: Communication between Job Management Service and the cluster

The Job Management service provides methods to handle job submission and file transfer. All operations done on the cluster are done under a generic grid account, but files and jobs are registered with the session ticket used for authentication, and so are associated with a user account present on the Single Sign-On service. This allows the handling of basic rights for users and the management of the lifetime of objects.

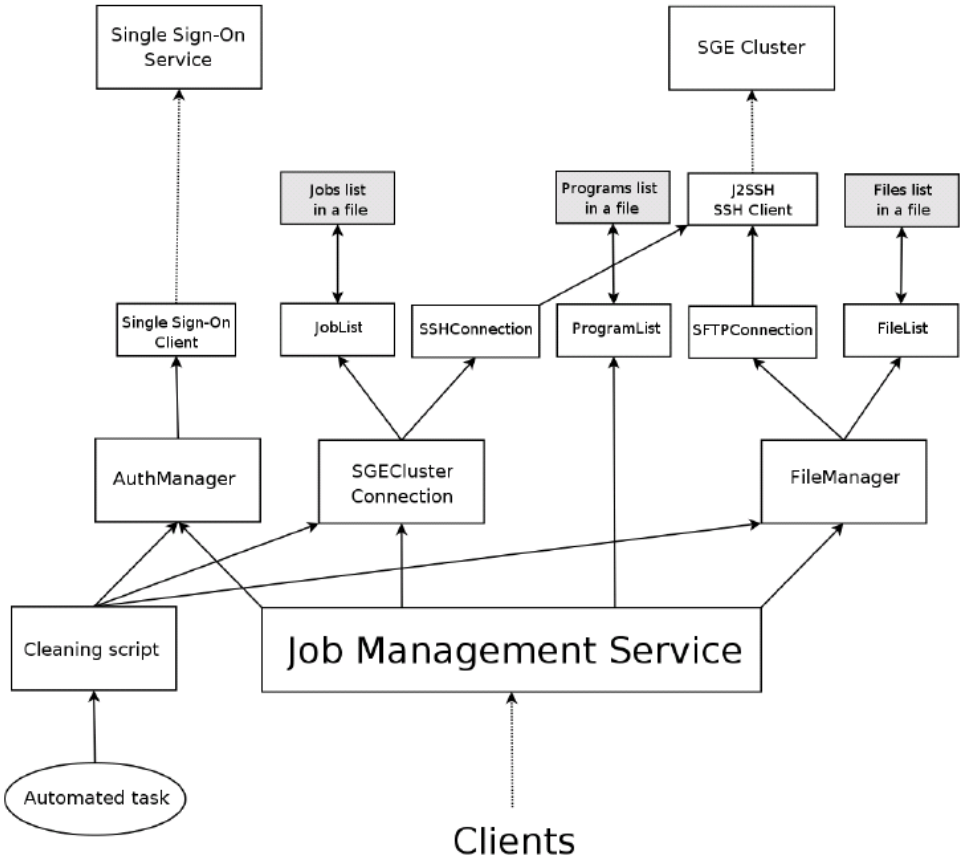

Figure 4: Architecture of the Job Management Service

4.1.1 Files

The service allows users to upload files to the cluster. This allows users to provide input files to a program, such as meshes for CAE software. Users can also download files resulting from the execution of a program. Files are stored on the cluster under the grid account, but every virtual user has his own directory. A list of all the files and their status is kept by the web service in order to avoid problems such as overwriting an existing file. Three file manipulation methods are provided: **UploadFile**, **DownloadFile**, and **DeleteFile**. A file has the same lifetime as the session. This means that when the authentication ticket expires, files are deleted. However, they can be kept if the user extends them by calling the **ExtendFile** method.

4.1.2 Jobs

Jobs, in the form of SGE batch scripts, are submitted to the cluster using the **SubmitJob** method. After job submission, **GetStatus** can be used to check whether the job is queuing, transferring, running or done and **KillJob** can be used to kill the job if the user does not need it anymore. There is also a **CleanSession** method that can be called to clean the user directory by killing all running jobs and removing all output and input files.

Two final methods are provided by this service: **CreateProgram** allows an uploaded file to be recognized as a script submitting jobs, and **DeleteProgram** removes a program from the list, but only one belonging to the user.

The internal architecture of the service is shown in Figure 4. It includes a script run automatically to remove files whose lifetime has expired.

4.2 Open Loop Repository service

This service stores all the results from the open loop tests. It provides clients a way to search in the list and to insert new elements in the list. It was decided to use a relational database to store the open loop results. Each industrial process supported by the system has its open loop results stored in a specific table. Every input or output of the process is a field of the table. The date of insertion of the record is also added as another field for testing reasons. The key of the table is defined with all the input fields to avoid having two identical situations present in the table.

This service provides two methods for the clients. The first one is **Insert**, which inserts an (Inputs,Outputs) pair into the table corresponding to the process identified in the arguments. The second method corresponds to a database search. It is named **getClosestValue** and searches in the table corresponding to the process for the record having the

inputs closest to the one given in the arguments. The notion of closest is defined by considering the following distance function.

We consider an industrial process having n inputs and m outputs. $I_i(k)$ is the i^{th} input of the k^{th} record. D_i is the i^{th} of the desired inputs.

$$Distance(I(k), D) = \sum_{i=1}^{i=n} \mid D_i - I_i(k) \mid \quad (1)$$

The record returned will be the one having the smallest distance from the desired inputs. What is exactly returned is a vector containing: the smallest distance, the inputs corresponding to this distance and the corresponding outputs of the record chosen. If the distance is zero, it means that the exact inputs have been found. If the distance is nonzero, we have an approximation of the desired situation.

4.3 Process Control Service

This service provides only one method, called **getOpenLoopData**, which takes process inputs as parameters and returns process outputs. When the client calls this method, the Process Control Service then calls the Open loop Repository Service to check whether such a situation is present in the database, by calling the **getClosestValue** method. If it is the case, i.e. when the distance returned is null, the Process Control Service simply gives the outputs received from the Open Loop Repository back to the client.

If the required distance in nonzero, the Process Control service will also return to the client the approximated data it received, but it will also create a new thread. The data are given back to the client so that the controller can continue to control the industrial process in real time. The newly created thread works in the background to fill the database with a new record corresponding to the situation, so that if the same situation is faced again, exact data will be available. To do this, the thread calls the Job Management Service **SubmitJob** method to launch a new open loop simulation, wait until it is done by checking at regular intervals using the **getStatus** method, download the results and then call the Open Loop Repository service to have the data inserted in the database.

4.4 Single Sign-On Service

This service handles authentication and session management using session tickets. Before performing actions on the grid, a user must obtain a ticket, which is represented here by a random integer and has a limited lifetime. The ticket can be extended if the operations the user has to make on the grid last longer than the original lifetime. When all the actions a user has to make on the grids are over, a user can destroy its session ticket to avoid any risk of having it reused by someone else.

As a call to any service is made with a session ticket as argument, all other services must have a way to authenticate these tickets. The single sign-on service provides methods to validate a ticket and obtain its owner.

4.5 Client

The client is the only part of the architecture that is not built with the objective to be kept as general as possible. It is because the client will have to talk directly with process control software, and for any kind of software a specific client will have to be built. Moreover as soon as the open loop data are received by the client, the manipulations made on these data depend on the process itself and on the algorithm used for process control.

However one part of the client is reusable. It corresponds to the classes handling communications with the Single Sign-On service (for session creation) and the process control service (to get open loop data). These actions are made available by creating an instance of a class called **GridServicesHandler**, and by using the local methods it provides. Using this, users will be able to build their own client for a particular industrial process, following the scheme represented in Figure 5.

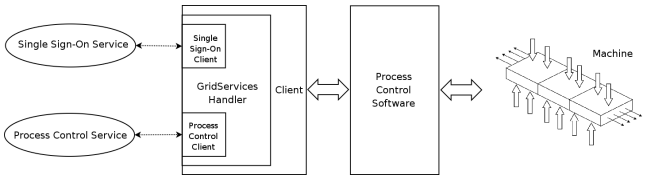

Figure 5: Relations between client, web services and process control software

5. IMPLEMENTATION AND TESTS

Our system was implemented on 3GHz Pentium 4 computers running Linux. The application server used was Jakarta Tomcat [2], with Apache Axis[3] as the web services framework. The entire application was programmed in the Java language. The database system used is MySQL[8], and for the ssh connexion to the cluster we used J2SSH java ssh client[9]. All timing results shown are the median of 10,000 experiments, unless otherwise indicated.

Previous research has already shown that coupling CAE-generated open loop tests with MPC leads to an improvement in process control[6]. This is why we do not focus here on quantifying this improvement, as it depends on the process itself. Tests were made using a simple industrial process we designed and modeled. We focused on measuring the overhead of the grid architecture designed. To make comparison possible, we also tested every case with a modified version, where the data management code from the open loop repository service is directly plugged to the process controller, as if the database search was done locally by the process controler. Calculating the difference between a normal call and this one gives the overhead of the grid architecture.

The sequence of operations in a call to the Process Control Service is shown in Figure 6, together with the timing of each operation. These times were obtained with all the services present on a single machine, to measure precisely the overhead caused by the use of web services. Notice that most of the overhead is caused by the transport, which includes encoding and decoding to and from SOAP messages. These

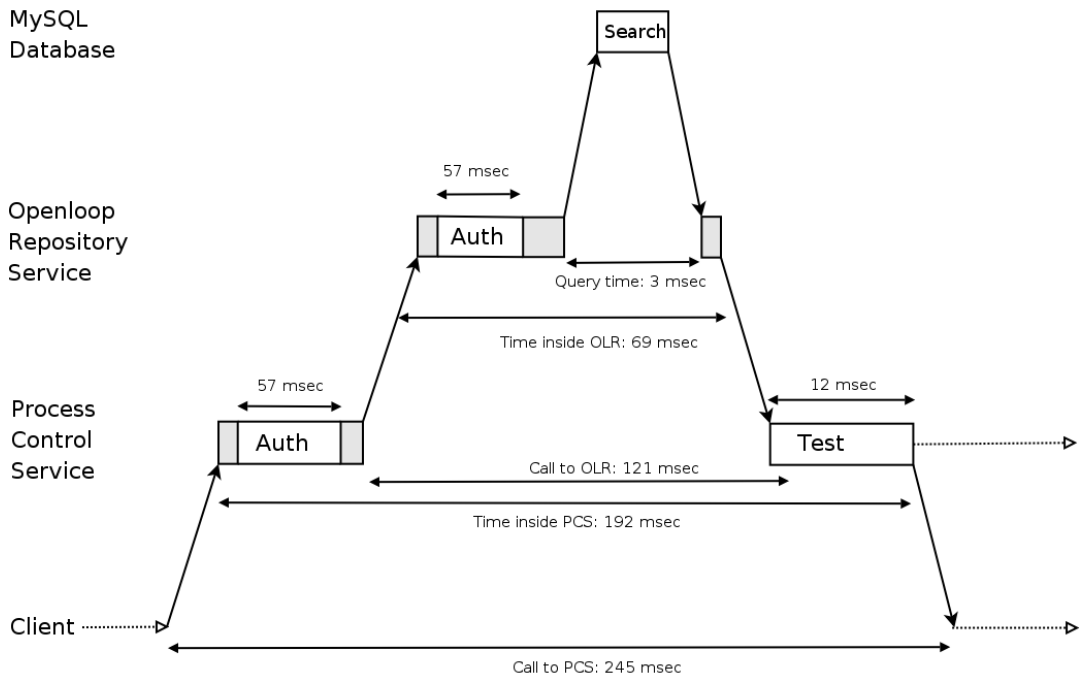

Figure 6: Sequence of operations in a call to the Process Control Service

calls are done between the client and the Process Control Service, and between the Process Control Service and the Open Loop Repository service. There is also communication involved in the ticket verification step, as it involves a call to the Single Sign-On service each time any method of any other service is called. A ticket verification takes 57 milliseconds, of which transport over the local loop represents 52 milliseconds. The overhead involved in other parts of the application, such as loading the configuration, checking the arguments given or storing states in files, is very small compared to the time spent in transport. Given these elements, the time for the client to call the getOpenLoopData method of the Process Control service and get the desired data can be broken down as follows:

$$\begin{aligned} T_{getOpenLoopData} &= T_{TransportClient\leftrightarrow PCS} \\ &+ T_{TransportPCS\leftrightarrow OLR} + T_{TransportPCS\leftrightarrow SSO} \\ &+ T_{TransportOLR\leftrightarrow SSO} + 2T_{InsideSSO} + T_{InsidePCS} \\ &+ T_{InsideOLR} + T_{Query} \end{aligned} \quad (2)$$

$T_{InsidePCS}$, $T_{InsideOLR}$ and $T_{InsideSSO}$ were found to be respectively 14 ms, 9 ms and 5 ms with the hardware and software configuration detailed above. As the calls to the getOpenLoopData method of the Process Control Service and to the getClosestValue of the Open Loop Repository service have the same arguments and output types, messages will be of same size. This leads us to conclude that with the same network conditions, $T_{TransportClient\leftrightarrow PCS}$ is equal to $T_{TransportPCS\leftrightarrow OLR}$.

The time to query the database, which corresponds to the time taken to get the data if it is directly plugged to the process controller is quite short (3 milliseconds) in the case where only a single record is present in database. The query corresponds to a multi-dimensional search, on a list of (input, output) pairs. In our test case there are four reals as inputs and three time series (with 300 elements each) as outputs. Table 1 shows that the query time remains modest as the number of records increases, reaching 2 seconds with 400 000 records.

Records in DB	DB size	Direct query	Query through grid architecture
1	$<$ 1MB	3 ms	245 ms
400 000	11GB	1962 ms	2220 ms

Table 1: Influence of database size on query times

To test more realistic cases, where services are deployed on different machines of the same type, we first measured separately each transport time, with 1000 tests in different traffic conditions, using two identical machines. These results are given in Table 2. The difference between Client-SSO transport and Client-PCS transport is caused by the different size of the data. Where the SSO only returns a ticket and a date, corresponding to two integers, the PCS has to return three time series of 240 reals each. We then used Equation 2 to deduce the total time needed to get open loop data. One condition for this approach to be valid is that traffic from different calls does not collide. This is not the case as the data transferred have limited size and the bandwidth used remains a small portion of the available bandwidth on the shared resources.

Once these individual times have been measured, we can compose some representative situations and calculate the

Resources between client and server	Round trip ping time	Transport Client-SSO	Transport Client-PCS
Local loop	0.04 ms	52 ms	53 ms
Single Switch	0.5 ms	53 ms	54 ms
Switch - Router - Switch	1.5 ms	62 ms	89 ms
Internet*	120 ms	582 ms	727 ms
Internet + DSL line to client**	300 ms	1341 ms	1787 ms

Table 2: Influence of network between services on running time (* ENSM.SE (France) - UNB (Canada); ** Aliant DSL - UNB)

time to get open loop data in each case. Four situations have been selected, and results are shown in Figure 7. For each situation the value presented is the time for the client to call the Process Control Service and get open loop data, with a database containing only the default open loop test. The second and third cases appear to be the most realistic ones for our purpose, illustrating the case where all the services are deployed within a manufacturer's intranet, and the computer controlling the machine could be on the same intranet or accessible remotely via the Internet. In the latter case the overhead involved is within a range of a few seconds, which is comparable to the time to query a large dataset.

6. CONCLUSION

This study proposed a grid architecture to solve the problem raised by the dynamic use of open loop simulation to improve the control of industrial applications. The architecture designed has been implemented using existing web services standards without using over-powerful but complex tools provided by the grid services framework. All the drawbacks of using pure web services have been worked around at the implementation level.

Our tests reveal that the overhead involved in encoding and decoding by the use of web services is approximately 250 ms, when all services are deployed on an intranet. This is well within acceptable limits for the intended use of GridMPC, such as the control of injection molding machines. This limited overhead is one of many key features of GridMPC, such as the reusability of services and the possibility of controlling many industrial processes at the same time.

A more significant limit to the performance of GridMPC is the query time of the database, which in our implementation has linear complexity. In practice, a tradeoff must be found between query time and suitability of approximate open loop results. Further work will involve exploring better algorithms for multidimensional searching, and the addition of a process monitoring service to allow improved parameters to be provided to the CAE simulations.

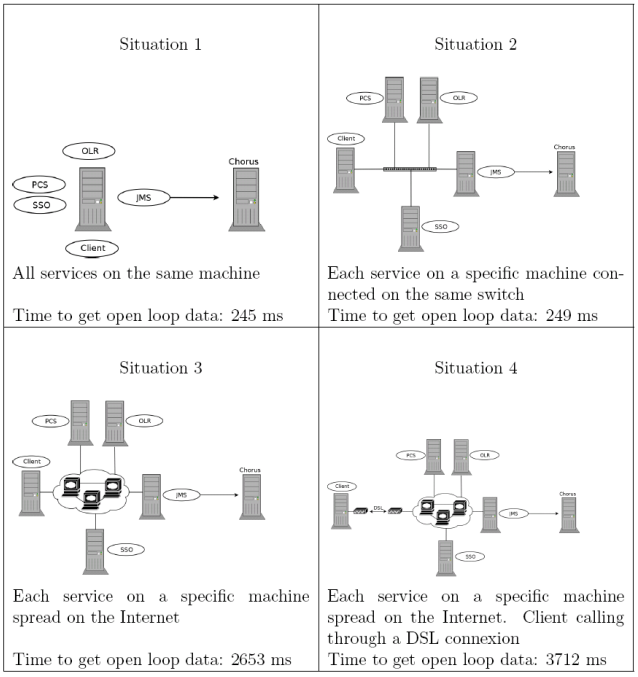

Figure 7: Description of the network architectures considered

7. ACKNOWLEDGEMENT

This work was funded by a Discovery grant of the Natural Sciences and Engineering Research Council of Canada.

8. REFERENCES

[1] Advanced Computing Research Laboratory. http://acrl.cs.unb.ca.

[2] Apache Foundation. The Apache Jakarta project - Apache Tomcat. http://jakarta.apache.org/tomcat.

[3] Apache Foundation. Web Services - Axis - User's guide. http://ws.apache.org/axis/java.

[4] M. Atkinson, D. DeRoure, A. Dunlop, G. Fox, P. Henderson, T. Hey, N. Paton, S. Newhouse, S. Parastatidis, A. Trefethen, and P. Watson. Web service grids: An evolutionary approach. *Concurrency and Computation: Practice and Experience*, 17:377 – 389, 2005.

[5] R. Bramley, K. Chiu, J. C. Huffman, K. Huffman, and D. F. McMullen. Instruments and sensors as network services: Making instruments first class members of the grid. Technical report, Indiana University, 2004.

[6] A. G. Gerber, R. Dubay, and A. Healy. Cfd-based predictive control of melt temperature in plastic injection molding. *Applied Mathematical Modelling*, 30(9):884–903, 2006.

[7] Globus. The Globus Toolkit. http://www.globus.org.

[8] MySQL AB. Mysql database. http://www.mysql.com.

[9] R. Pernavas. J2SSH API. http://freshmeat.net/projects/sshtools-j2ssh.

[10] P. D. Roberts. A brief overview of model predictive control. In *IEE Seminar on Practical Experiences with Predictive Control (Ref.No.2000/023)*, pages 1/1–1/3, 2000.

[11] J. Rosenberg and D. Remy. *Securing Web Services with WS-Security*. SAMS Publishing, 2004.

[12] P. Stodghill, R. Cronin, K. Pingali, and G. Heber. Performance analysis of a multi-physics simulation system based on web services. preprint: http://www.cs.cornell.edu/stodghil/papers/grid03.pdf.

[13] Sun Microsystems. Sun Grid Engine Project. http://gridengine.sunsource.net.

[14] G. Wasson and M. Humphrey. Exploiting WSRF and WSRF.NET for remote job execution in grid environments. In *Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium*, 2005.