Security mechanisms for legacy code applications in GT3 environment.

Gabor Terstyanszky[1]
Thierry Delaitre[1]
Ariel Goyeneche[1]
Tamas Kiss1
K. Sajadah[1]
Stephen Winter[1]
Peter Kacsuk[1,2]

[1]Cavendish School of Computer Science, University of Westminster
[2]MTA SZTAKI Lab. Of Parallel & Distributed Systems, Budapest, Hungary

Security Mechanisms for Legacy Code Applications in GT3 Environment

G.Terstyanszky[1] T. Delaitre[1], A. Goyeneche[1], T, Kiss[1], K. Sajadah[1], S.C.Winter[1], P. Kacsuk[1,2]
[1]*Centre of Parallel Computing,Cavendish School of Computer Science,*
University of Westminster, 115 New Cavendish Street, London W1W 6UW,
[2]*MTA SZTAKI Laboratory of Parallel and Distributed Systems*
H-1518 Budapest, P.O. Box 63, Hungary
e-mail: testbed-discuss@cpc.wmin.ac.uk

Abstract

There are many legacy code applications that cannot be run in Grid environment without significant modifications. To avoid re-engineering of legacy code, we developed the Grid Execution Management for Legacy Code Architecture (GEMLCA) that enables deployment of legacy code applications as Grid services. GEMLCA is an OGSI Grid service layer that supports submitting jobs, getting their results and status back. Security requirements are essential to any Grid application to preserve the confidentiality and integrity of data. To meet these requirements the GT3 security model was implemented in GEMLCA. The paper introduces GEMLCA and how Grid Security Infrastructure (GSI) components have been added to GEMLCA in order to enable secure execution of jobs in Grid. The paper also presents how a legacy code traffic simulator was transformed into a Grid service using GEMLCA and gives some simulation results.

1. Introduction

Legacy code applications may need additional compute resources as a result of the amount of data to be processed or tasks to be performed. If additional compute resources are not available, it could be too expensive to purchase and maintain them. To avoid this situation, legacy code applications can be deployed and executed in a Grid environment to provide access to additional compute resources. In order to run legacy code applications on the Grid, the program can be either re-engineered or offered as a Grid Service without any re-engineering using Grid Execution Management for Legacy Code Architecture (GEMLCA).

GEMLCA was developed to support deployment of legacy code applications as Grid services without modifying or even requiring access to the original code. GEMLCA must offer a secure environment to run legacy code applications as Grid services enabling only authenticated users to submit service requests to GridServices. To provide GEMLCA security the security mechanisms of Globus Security Infrastructure (GSI) of Globus Toolkit 3 (GT3) was used. Particularly, applications security is guaranteed at client-, message- and server-level using GSI in GEMLCA applications.

Chapter 2 analyses existing solutions to run legacy code applications in Grid environment. Chapter 3 describes GEMLCA introducing its architecture and how it works. Chapter 4 gives an overview how security is implemented in GT3. In Chapter 5 we explain how security was realised in GEMLCA. Chapter 6 introduces the implementation of GEMLCA. In Chapter 7 we present some simulation results.

2. Using Legacy Code in Grid Environment

Many large industrial and scientific applications are available today that were written well before Grid computing or service-oriented approaches appeared. To integrate these legacy code programs into service-oriented Grid architectures with the smallest possible effort and the best performance, is a crucial point in more widespread industrial take-up of Grid technology.

There are several research efforts aiming at automating the transformation of legacy code into a Grid service. Most of these solutions are based on transformation of legacy code applications into Web services outlined in [1], and use Java wrapping in order to generate stubs automatically. One example for this is presented in [2], where the authors describe a semi-automatic conversion of legacy C code into Java using Java Native Interface (JNI). After wrapping the native C application with the Java-C Automatic Wrapper (JACAW) MEdiation of Data and Legacy Code Interface tool (MEDLI) is used for data mapping in order to make the code available as part of a Grid workflow.

Different, non-wrapping approaches are presented in [3] and [4] but these solutions only define the principles of legacy code transformation and do not specify an environment or a tool to do the automatic conversion.

The work presented in this paper is supported by the "Proposal to Evaluate OGSA/GT3 on a UK Multi-site Testbed" EPSRC project (Grant No: GR/S77509/01

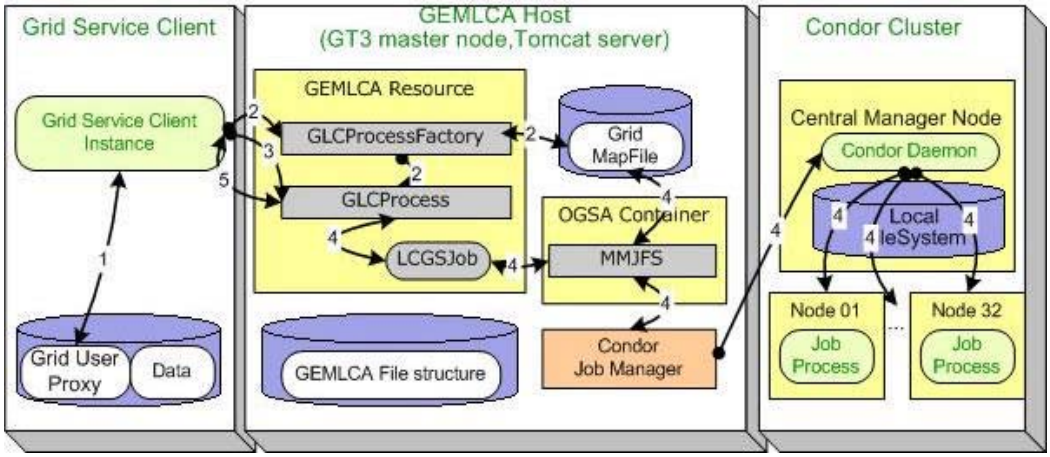

Figure 1. GEMLCA Lifecycle Management

Compared to Java wrapping GEMLCA is based on a different principle. It offers a front-end Grid service layer that communicates with the client in order to submit service requests, manage input and output parameters, and contacts a local job manager through Globus MMJFS (Master Managed Job Factory Service) to submit the legacy computational jobs. To deploy a legacy application as a Grid service there is no need for the source code and not even for the C header files as in case of JACAW. The user only has to describe the legacy parameters in a pre-defined XML format. The legacy code can be written in any programming languages and can be not only a sequential but also a parallel MPI or PVM code that uses a job manager like Condor and where wrapping can be difficult.

3. GEMLCA

GEMLCA is an architecture that supports deployment of legacy code applications as Grid services without re-engineering the original code. GEMLCA is based on OGSI [5] and GT3 [6] infrastructure. However, the concept of GEMLCA is more generic and can also be applied to other service-oriented architectures. Using different platforms the communication and the actual service implementation is different, but the concept of the architecture remains the same. This way the transition to new emerging standards like Web Services-Resource Framework (WSRF) and GT4 will be straightforward.

GEMLCA has been designed as a three-layer architecture: the first layer, the *front-end* layer, offers a set of Grid Service interfaces that any authorized Grid client can use in order to contact, run, and get the status and any result back from the legacy code. This layer hides the second layer, the *core* layer, which deals with each legacy code environment and their instances as Grid legacy code processes and jobs. The final layer, the *back-end* is related to the Grid middleware where the architecture is being deployed. All three GEMLCA layers were developed by the Centre of Parallel Computing, University of Westminster. The GEMLCA environment uses either Globus Fork or Condor [7] as a job manager. To utilise GEMLCA with other Grid middleware than GT3, like GT4 or a "pure" Web Services based approach, only the back-end layer has to be modified.

A GEMLCA resource is composed of a set of Grid services that provides a number of Grid interfaces in order to control the life-cycle of the legacy code execution. This architecture can be deployed in several user containers or Tomcat application contexts.

In order to access a legacy code program, the user executes the GEMLCA Grid Service client which creates a legacy code instance with the help of the legacy code factory. Following this, the GEMLCA resource submits the job to the compute server through GT3 MMJFS using a particular job manager.

Figure 1 presents the GEMLCA implementation and its lifecycle. The scenario for submitting legacy code jobs using the GEMLCA architecture is composed of the following steps:

(1) The user signs his/her certificates to create a Grid user proxy. The proxy contains the user's Grid credential to be delegated by the GEMLCA Grid services to MMJFS for the allocation of resources.

(2) The Grid Service Client, using the Grid Legacy Code Process Factory (GLCProcessFactory), creates a Grid Legacy Code Process (GLCProcess) instance where the initial process legacy code environment is set and created using the GEMLCA file structure.

(3) The Grid Service Client sets and uploads the input parameters needed by the legacy code program exposed by the GLCProcess and deploys a job using a Resource Specification Language (RSL) file and a multiuser/instance environment to handle input and output data.

(4) If the client credential is successfully mapped, MMJFS contacts the job manager that allocates resources and executes the parallel legacy code in a computer cluster.
(5) As far as the client credentials are not expired and the GLCProcess is still alive, the client can contact GEMLCA for checking job status and retrieve it at any time.

Finally, when the Grid Service instance is destroyed, the multi-user/instance environment is cleaned.

The description of the GEMLCA class structure and implementation of its classes are given in [8].

4. GT3 and Security

The Globus Toolkit uses GSI [9] to enable authentication, to implement authorization and to support secure communication over a computer network. GT3 provides a set of security components based on GSI to implement various security mechanisms in Grid applications. GSI provides three levels of security for Grid applications: server-side security, message-level security and client-side security.

The client-side security is based on authentication, authorisation and credential delegation. A user can use all or a subset of these security measures before sending a service request.

Users have to create Grid user proxies to authenticate themselves. They sign their certificates while they are creating their Grid user proxies. The user proxy enables single sign-on in Grid environment. Signing the service request means that the user's Grid certificate and signature are attached to the request. Users can use either the Secure Message handler or the Secure Conversation Message handler to sign/encrypt service requests.

The user can also configure the authorization mechanism. Authorisation is used to determine when a user has a right to access a resource and perform a job. GSI manages authorization by reading the user's identity (or identity certificate) from the user proxy and maps this identity to a local identity. Currently, three authorization methods are supported by GSI: *none*, *self*, and *grid-map*. The *"none"* mechanism implies that authorization will be disabled on the server side. The *"self"* mechanism means that only clients with the same identity as the service are allowed to access the Grid service. The *"grid-map"* mechanism implies that grid-map file authorization is performed.

The credential delegation enables a Grid service to utilise the user's credential in order to invoke other services on the user's behalf. Therefore, a Grid service is able to set the delegated credentials as its identity before contacting other Grid services. Credential delegation with user proxy supports single sign-on in Grid applications. GSI provides different delegation modes: *full proxy delegation*, *limited proxy delegation* or *no delegation*. To activate the credential delegation the service stub must be configured. To complete the delegation some security code must be also added at the beginning of all Grid services' methods. This code enables the Grid service to check the user's identity in order to authenticate it to other Grid services. So, when a Grid service receives a user's request, it forwards it to the MMJFS. The MMJFS identifies the client who sent the request.

GT3 offers two message-level security modes: GSI Secure Session mode and GSI Secure Message mode [10]. The GSI Secure Session approach creates a security context with the server before requesting a Grid service. In contrast, the GSI Secure Message approach does not create a security context before sending a message. The message-level security uses WS Security, XML Encryption and XML Signature standards to provide security between Grid clients and Grid services.

To implement the message-level security and to send requests to Grid services the Secure Conversation Service handler, the Secure Message handler and the Secure Conversation Message handler are used as client-side security handlers. A user has to pass information to the client-side handlers on what type of security to use. In the Secure Session Mode the Secure Conversation Service handler establishes a security context (or session) through a secure conversation assigned to the Grid service to which the client wants to communicate. The handler ensures that a security context is established whenever it gets a message from the client indicating that session-based security is required. After establishing the secure session the handler passes the client's request to the Secure Message handler. The handler encrypts and/or signs the message with user's credentials using the security context. In the Secure Message mode the Secure Conversation Message handler's task is to sign and/or encrypt messages. GSI uses the WS-Security client handler on the client side to receive responses from Grid services. The main task of the WS-Security client handler is to verify and decrypt any encrypted and/or signed messages.

At the server-side when a request arrives for a Grid service, the WS-Security handler, the Security Policy handler, the Authorization handler, the Secure Conversation Message handler and the Secure Message handler may be invoked to check the security information of the request before forwarding it to the Grid service. The first three handlers manage incoming service request while the last two handlers deal with outgoing responses. First, the WS-Security handler checks whether the request is encrypted or signed. Secondly, the Security Policy handler controls whether the request meets the security requirements of the service. Finally, the Authorisation handler checks whether the client is authorized to invoke the service. If the request has successfully passed all three handlers, it is sent to the requested Grid service for processing. In the GSI Secure Session mode the Secure Conversation Message handler encrypts and/or signs messages using the established security context before

returning the reply. In the GSI Secure Message mode the Secure Message handler encrypts and/or signs messages before sending the response.

The server-side security is provided through service authorisation and credential management. The security is specified via the security deployment descriptor and the security configuration file. The security deployment descriptor defines how to configure authentication methods and run-as identities to access Grid services. It is loaded when a Grid service is activated. The security configuration file specifies how each method of a Grid service must be accessed. The specification is given by an XML file and a pointer should be added to the deployment descriptor to point to this file.

5. GEMLCA Security Model

GEMLCA legacy code applications should be run as Grid services allowing only authenticated users to submit service requests and authorised users to access compute resources. To achieve the required security the client-side should incorporate authentication, authorization and credential delegation. The server-side security is implemented through service authorization, service credentials and credential delegation. Figure 2 presents the GEMLCA security model.

The GEMLCA Resource must be configured with service credentials given in the security configuration file to assign identity to Grid services. GEMLCA also uses the security configuration file to define how methods of Grid services must be accessed using authentication and how methods to be run with the user's security identifier. GEMLCA utilises the grid-map authorisation mechanism in order to control access to Grid services. When a service request reaches a Grid service, the service request handler checks the grid-map file to control if the user is authorised to access the Grid service. A client is authorized to access the Grid service if it has an entry in the grid-map file.

The user has to set up the authentication and authorisation modes. To ensure that the service is executed on the user's behalf, credential delegation mode must be also activated at the client side. The GEMLCA Client has to sign its credential with its certificate and set up full delegation mode. To do it a user must create a Grid user proxy that makes the user's credentials available in calls to the Grid service factory. The credential tells the Grid service that the user allows using its credential to invoke other services on its behalf. Thus, the Grid service will be able to use the delegated credentials as its identity before contacting other services. Through proper security configuration the credential will be delegated by the Grid service to MMJFS for resource allocation.

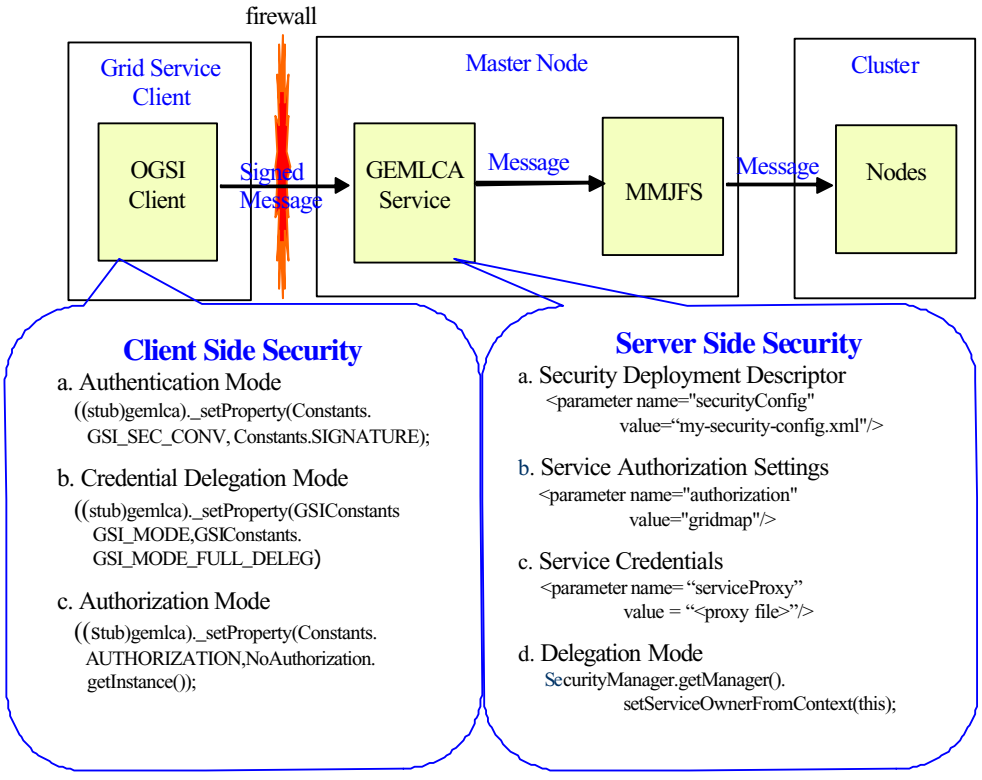

Figure 2. GEMLCA Security Model

On Fig. 2 the client generates a Grid user proxy signing it with its certificate. The proxy is used to authenticate the user to a Grid service. Having the proxy, the client generates a service request describing the job to be executed on its behalf and signs the request with its proxy.

If the user wants to ensure privacy, it can encrypt the service request. The user sends the signed and/or encrypted service request as a message to the GEMLCA Service Factory. To increase the security a firewall is installed between the Grid Client and Grid Service Factory.

When a Grid service receives a service request first, it verifies the request's signature to authenticate the user. Secondly, the Grid service checks if the user is authorised to access the Grid service. If the user has the required authorisation and the credential delegation mode is set, the job request is forwarded to MMJFS. The MMJFS submits the job to the compute resource, for example to a cluster, where it is executed on behalf of the user.

6. GEMLCA Implementation

The GEMLCA is implemented by deploying a set of Grid services, which represent GEMLCA Resources, and tested by using secure Grid clients.

The Grid client is a Java program executed by the Java Virtual Machine from a Grid portal based on P-GRADE Grid portal. The GEMLCA Resource and MMJFS are deployed in two separate Java servlet engine containers, particularly in Tomcat web application contexts hosted by a single Tomcat server running on the Westminster GT3 master node.

The architecture presented in Chapter 3 requires a specific job manager such as Fork, Condor or Sun Grid Engine to be configured for submitting computational jobs to clusters. Condor is selected as the job management facility for the Westminster cluster and it requires the Condor job manager interface to be installed and configured as well as the GT3 master node to be configured as a submit host to the Parsifal Condor pool. The default installation of GT3 only installs the Fork job manager and an additional step is required to install and configure Condor, which is bundled with GT3.

GEMLCA uses GSI to enable user authentication and to support secure communication over a Grid network. A GEMLCA client needs to sign its credential and also to work in full delegation mode [11] in order to allow the GEMLCA environment to work on its behalf in order to

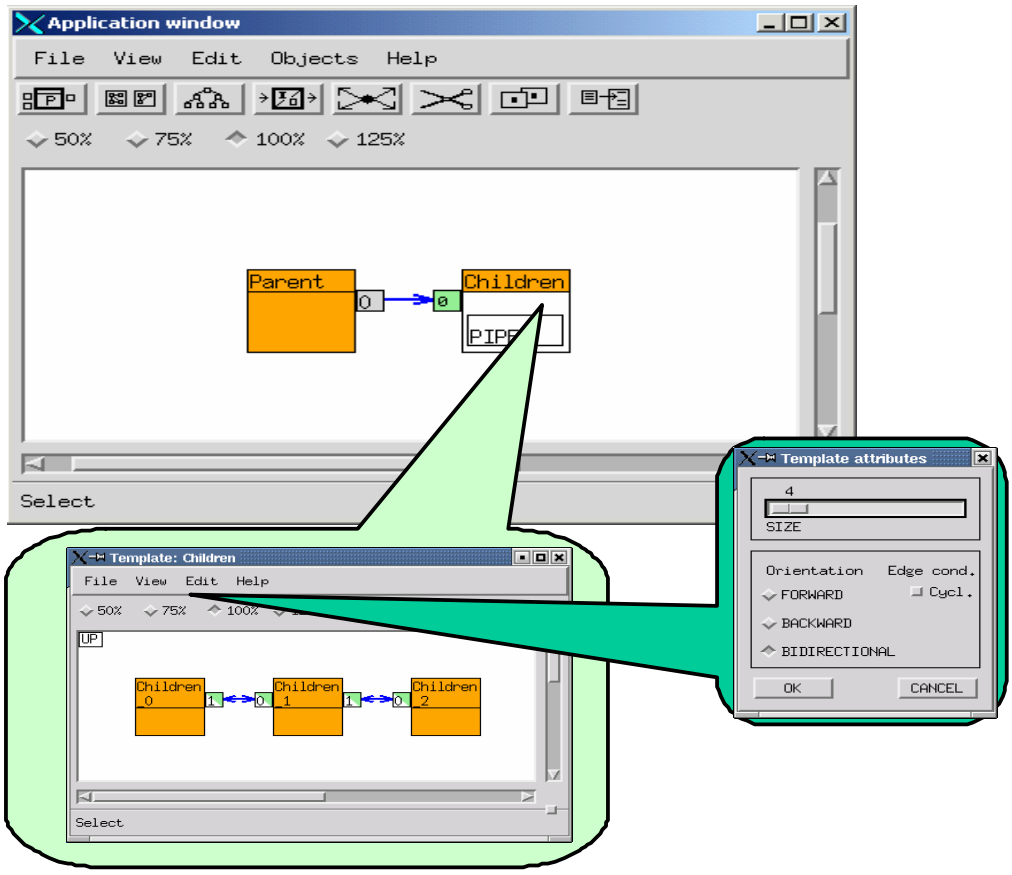

Figure 3. MadCity in P-GRADE environment

create the legacy code environment and pass the user's Grid credential to MMJFS.

GEMLCA has two levels of authorisation. The first level is implemented by the grid-map file mechanism. The second level is defined by a set of legacy codes that a Grid Client is allowed to use. The set is composed of a combination of a general list of legacy codes, available to anyone using a specific GEMLCA resource, and a user mapped list of legacy codes, only available to Grid clients mapped to a local user by the grid-map file mechanism.

GEMLCA administers the internal behaviour of legacy codes taking into account the requirements of input files and output files in a multi-user environment, and also complies with the security restrictions of the operating systems where the architecture is running. In order to that, GEMLCA is using GEMLCA itself in a protected mode composed of a set of system legacy codes in order to create and destroy a unique process and job stateful environment only reachable by the local user mapped by the grid-map file mechanism.

7. Traffic Simulation Using GEMLCA

The MadCity traffic simulator was used as legacy code application to be run as a Grid Service through GEMLCA. MadCity traffic simulator [12] was developed by the research team of Centre of Parallel Computing, University of Westminster.

MadCity simulates traffic on a road network and shows how vehicles move on roads and at junctions. It consists of the GRaphical Visualiser (GRV) and the SIMulator (SIM) tools. The GRaphical Visualiser helps to design a road network file. The SIMulator models the movement of vehicles using the road network file. After completing the simulation, the SIM creates a trace file, which is loaded on GRV in order to display the movement of vehicles.

The computational performance of the simulator depends on a number of parameters, such as number of vehicles, junctions, lane cut points and roads. The road network can contain thousands of vehicles, roads and junctions. SIM uses a simple set of rules to compute the new position and state of each vehicle taking into account its current position and the road network.

The SIM of the MadCity traffic simulator was parallelised using Parallel Grid Run-Time and Application Development Environment (P-Grade) [13]. A pipeline template was used where all nodes perform the same task simulating different segments of the road network. Four children nodes participate in the traffic simulation on Figure 3. The reason for using the pipeline template was its scalability; i.e. the number of nodes could be decreased or increased using the template attribute window without modifying the code.

To run MadCity as a Grid Service, the user has to create an XML-based Legacy Code Interface Description File (LCID) [14] that specifies the legacy code to be run, the job manager to be used, maximum/minimum number of job/processes and parameters required by the legacy code. The LCID file is added to the list of available legacy codes in the front-end layer. After adding the LCID file of the MadCity traffic simulator to the list of available Grid Services any Grid client, who has the required authorisation, can select and run the traffic simulator.

The performance results generated by the parallel version of MadCity traffic simulator as Grid Service in GEMLCA environment are given in Figure 4. The performance results are similar to performance results produced by the cluster-based version of the MadCity traffic simulator [15].

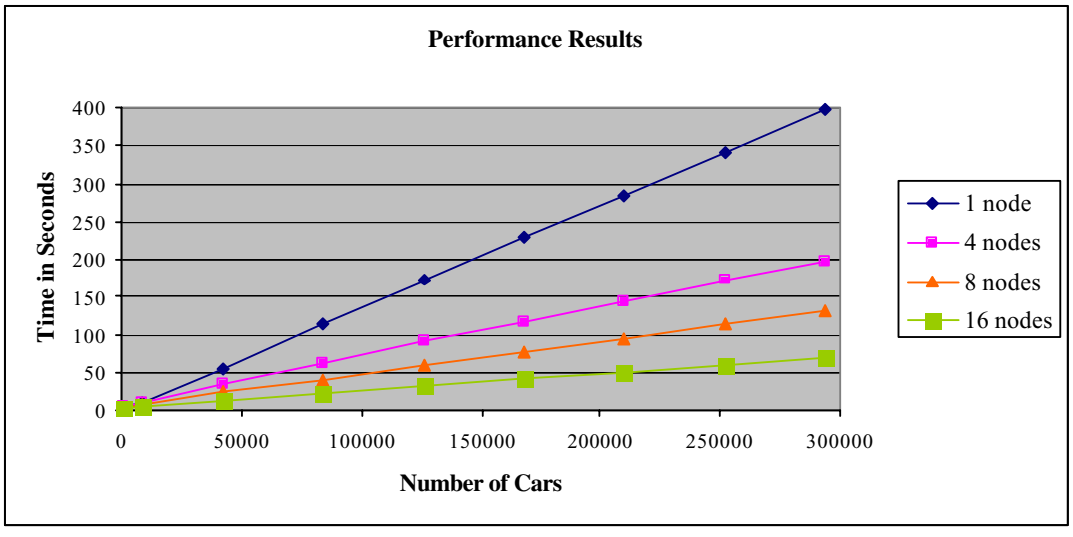

Figure 4. Simulation Results

8. Conclusions

GEMLCA environment was created to make existing legacy code applications available as OGSI Grid Services. The GEMLCA environment offers a set of OGSI compliant interfaces in order to create, run and manage Grid services that offer all the legacy code application functionality without changing the legacy code. GELMCA adds a software layer to existing Grid middleware like GT3 and provides an integrated Grid execution lifecycle environment for end users, such as chemists, meteorologists, etc.

GEMLCA security measures are based on GT3 GSI model. GEMLCA security offers single sign-on capability for submitting jobs, uploading input data and downloading output data using credential delegation and user's Grid certificate.

The parallel version of the MadCity traffic simulator was used as a legacy code application in the GEMLCA environment in order to test the architecture. It was proved that running the simulator as a Grid service did not require any re-engineering, only an XML based description file has to be created.

References

[1] D. Kuebler, andW. Eibach, Adapting legacy applications as Web services, IBM Developer Works.

[2] Y. Huang *et al.*, Wrapping Legacy Codes for Grid-Based Applications, Proceedings of the 17th International Parallel and Distributed Processing Symposium, workshop on Java for HPC), 22-26 April 2003, Nice, France. ISBN 0-7695-1926-1

[3] T. Bodhuin, and M. Tortorella, Using Grid Technologies for Web-enabling Legacy Systems, Proceedings of the Software Technology and Engineering Practice (STEP), Software Analysis and Maintenance: Practices, Tools, Interoperability workshop September 19-21, 2003, Amsterdam, The Netherlands.

[4] B. Balis, M. Bubak, and M. Wegiel, A Framework for Migration from Legacy Software to Grid Services, Cracow Grid Workshop '03, Cracow, Poland, December 2003.

[5] S Tuecke etal: Open Grid Services Infrastructure (OGSI) Version 1.0, June 2003 http://www.globus.org/research/papers/Final_OGSI_Specification_V1.0.pdf

[6] The Globus Toolkit 3 Programmer's Tutorial http://www.casa-sotomayor.net/gt3-tutorial/Grid Security ESSC, October 2002

[7] D. Thain, T. Tannenbaum, and M. Livny, "Condor and the Grid", in Fran Berman, Anthony J.G. Hey, Geoffrey Fox, editors, Grid Computing: Making The Global Infrastructure a Reality, John Wiley, 2003

[8] T. Delaittre, et al. Publishing and Executing Parallel Legacy Code Using an OGSI Grid Service, February 2004. Conf. Proc. of the 2004 International Conference on Computational Science and its Applications, Technical Session on Grid Computing, May 2004, Assisi, Italy, Applications. Editors: A. Lagana et al. LNCS 3044, pp. 10-19,

[9] GT3 Grid Security Infrastructure Overview, http://www-unix.globus.org/security/gt3-security-overview.doc

[10] Writing Secure Grid Services Using Globus Toolkit 3.0, October 2003 http://www.06.ibm.com/developerworks/grid/library/gr-secserv.html?Open&ca=daw-gc-dr

[11] GT3 Developers and Administrators Tutorial: Security , 2002 http://www.globusworld.org/globusworld-pre-8-5-03/globusworld_web/gt3/Session-7-GT3-Security.pdf

[12] MadCity Traffic Simulator http://www.cpc.wmin.ac.uk/madcity/madcity_report.doc

[13] P-GRADE User's Manual http://www.lpds.sztaki.hu/projects/p_grade/manual/manual_frame.html

[14] T. Delaitre, A. Goyeneche, P. Kacsuk, T. Kiss, G.Z.Terstyanszky and S.C. Winter, GEMLCA: Grid Execution Management for Legacy Code Architecture Design, Conf. Proc. of the 30th EUROMICRO conference, Special Session on Advances in Web Computing, August, 2004, Rennes, France, pp. 477-483

[15] A Gourgoulis, G. Terstyanszky, P Kacsuk, S C Winter, Creating Scalable Traffic Simulation on Clusters. PDP2004 Conf. Proceedings of the 12th EuroMicro Conference on Parallel and Distributed and Network-Based Processing, La Coruna, Spain 11-13th February, 2004,