ExcelGrid: A .NET Plug-in for Outsourcing Excel Spreadsheet Workload to Enterprise and Global Grids

Krishna Nadiminti, Yi-Feng Chiu, Nick Teoh, Akshay Luther, Srikumar Venugopal, Rajkumar Buyya[1]

Grid Computing and Distributed Systems Laboratory and NICTA Victoria Laboratory
Department of Computer Science and Software Engineering
The University of Melbourne
ICT Building, 111 Barry Street
Carlton, VIC 3053, Australia

***Abstract*: Most businesses today make use of spreadsheets for a lot of their daily activities. Spreadsheets are useful tools which enable quick analysis of data, and are within reach of every person. Microsoft Excel is a widely used spreadsheet application and provides a very easy-to-use system, which any user can utilise to perform complex analysis on data without having to learn programming. As businesses grow, the requirement for more and more complex compute intensive analysis arises and there is a need to improve the performance of the data analysis process. To meet this requirement, we propose the use of grids as they allow us to harness geographically distributed computational resources for performing computations in parallel in order to speed up the execution. This paper describes a method to grid-enable Microsoft Excel, to execute spreadsheet computations on enterprise and global grids.**

I. INTRODUCTION

Spreadsheets are powerful business analysis tools [6] as they are applicable in a wide range of areas including business, science, education, engineering etc. A spreadsheet is readily adaptable for problems that are iterative, recursive, or tabular in conceptual format and enables users to tinker with values of variables, constants, and step size and to explore the tempting "what if?" type of questions in the problem-solving process. Such parameter studies create numerous scenarios for exploration leading to an exponential increase in processing power requirements. These kinds of analyses take several hours or even days of processing time on desktop computers with regular spreadsheet applications. Such large-scale processing requirements can be met by peer-to-peer/desktop/enterprise [20] and global grids [17], as they allow harnessing computational resources such as desktops, servers, clusters which are distributed geographically.

Microsoft Excel is the leading spreadsheet application in markets all over the world today with a 90% market share [19] – not too long ago, Lotus 1-2-3 was considered the "standard" spreadsheet. Excel and spreadsheets in general, are very useful for applications in the business-financial domain such as risk and portfolio analysis. They also find applications in quantitative decision support systems in business planning. There are a wide range of add-ons to Excel, due to its popularity. These add-ins enable applications that include Monte-Carlo simulations and statistical analysis, time-series forecasting, real option analysis and the like Excel is also used in the engineering domain with MATLAB plug-ins. A grid-interface to Excel will help in extending this very useful software from the desktop to enterprise and global grids as they provide the benefits of improved job execution speed and result in getting processing done faster. However, an application needs to possess certain attributes to reap the benefits of grid-enabling it. For example, an application needs to be very resource and/or data-intensive. It should also be easily parallelisable. Ideally applications which can be divided into sub-tasks which can run independent of each other are well suited to run on the grid, if each of these tasks, are reasonably compute intensive such that the execution time of each task is much more compared to the network transfer overheads. Such coarse-grain applications are the ones which can gain maximum benefit from the grid. On the other hand, day-to-day spreadsheet applications involving trivial tasks, would only see a decline in performance, if run on a distributed system.

In this paper, we present an approach to grid-enable Excel by creating a plug-in, called ExcelGrid, which operates with grid systems, based on service-oriented architecture, such as Alchemi [12] and Gridbus broker [18] frameworks. The ExcelGrid plug-in provides a front-end to a grid via Excel spreadsheet and performs user-defined computations on enterprise grids created using Alchemi and global grids built using Gridbus coupled with Globus [7], UNICORE [11] and also Alchemi technologies. It allows users to run jobs on remote computers, using an easy-to-use GUI, and retrieve the results via the standard Excel spreadsheet interface.

The contributions of this work include a) the use of service oriented architecture (SOA) in design and development of an open-source grid plug-in for Excel, b) the use of modern technologies such as .NET, C#, web services and c) development of scalable ExcelGrid framework that transparently operates in both enterprise and global grid environments. In addition, we believe that the release of ExcelGrid as an open source software ensures its wider adoption and encourages other emerging Grid middleware developers to easily take advantage of it.

[1] Contact Author EmailID – raj@csse.unimelb.edu.au

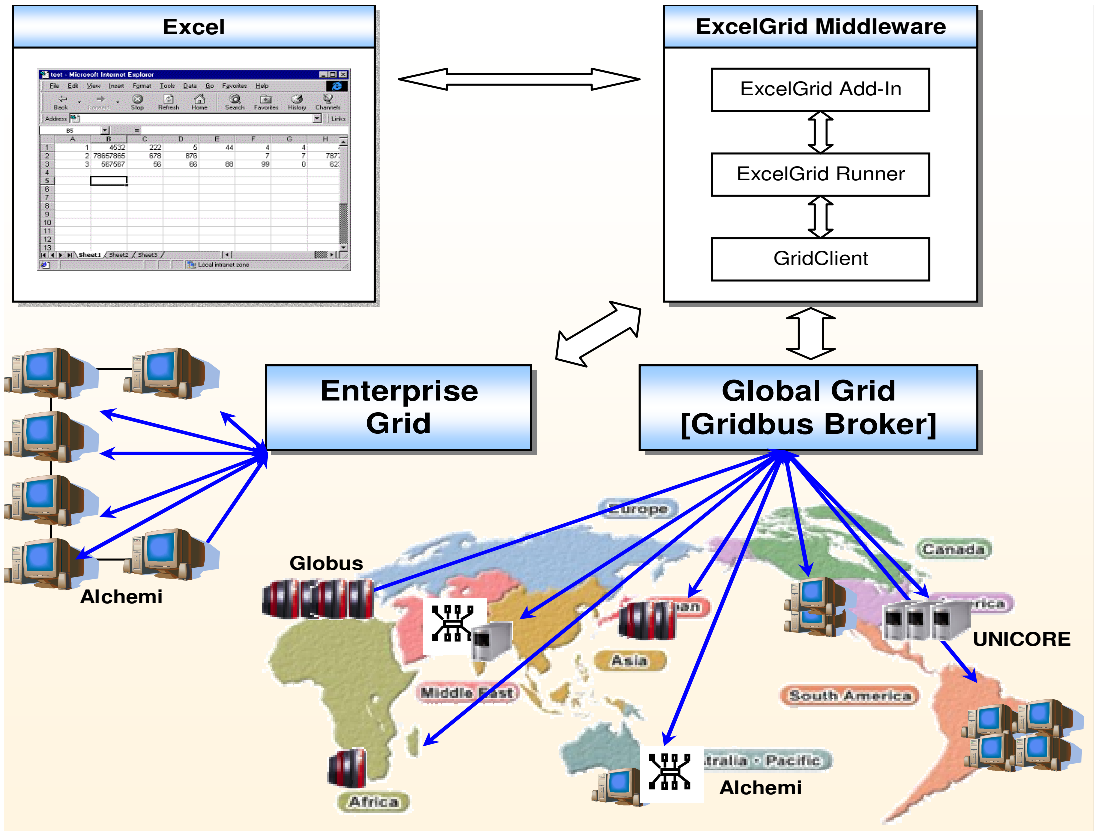


Figure 1: Excel Grid Architecture.

The rest of this paper is organised as follows. Section II presents grid technologies whose programming interface has been used in developing ExcelGrid plug-in. Section III introduces our proposal, ExcelGrid and its architecture. Section IV goes into further details of the design and implementation of our system and in section V performance results of ExcelGrid running on different grid middleware are presented. A review existing work in the area of enhancing the capabilities of Excel by providing extensions that enable execution of processes on a grid is given in section VI. We finally conclude in section VII with our plans for future work.

II. Grid Technologies

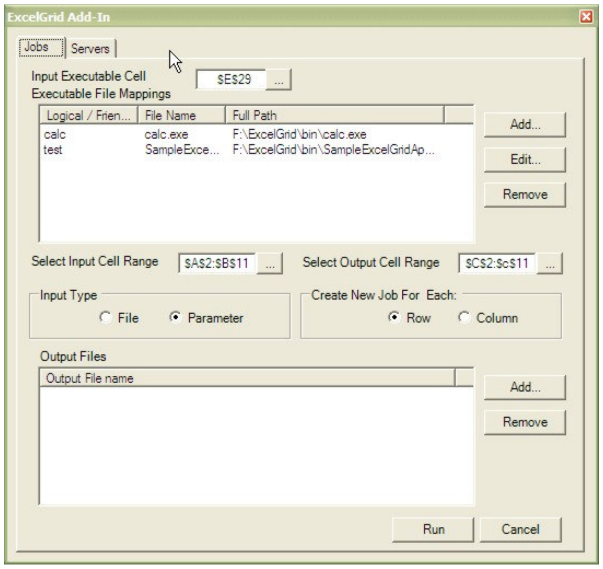
In the development of ExcelGrid we have used programming interfaces provided by Alchemi and Gridbus Broker for harnessing enterprise and global grids, we briefly describe these two technologies.

A. Alchemi and .NET framework

Alchemi [12] is a Windows-based desktop grid computing framework implemented on Microsoft .NET platform, and developed at the University of Melbourne as part of the Gridbus project, which provides the runtime machinery and the programming environment which is required to construct desktop grids and develop grid applications. It supports object-oriented grid application programming and execution of cross-platform applications via web services. The Microsoft .NET Framework is an ideal platform for grid computing middleware because it provides a powerful toolset that supports security, remote execution, multithreading, cross-language development, etc. The key features supported by Alchemi, are internet-based clustering of desktop computers without a shared file system, federation of clusters to create hierarchical, cooperative grids, dedicated or non-dedicated (voluntary) execution by clusters and individual nodes, object-oriented grid thread programming model (fine-grained abstraction), and a web services interface supporting a grid job model (coarse-grained abstraction) for cross-platform interoperability (e.g. for creating a global and cross-platform grid environment using a custom resource broker component).

B. Gridbus Broker and Global Grids

The Gridbus data-service broker is a platform-independent brokering framework, implemented in Java, which provides brokering services for execution of jobs on various low-level middleware systems like Globus, UNICORE. It hides the complexity of the grid by translating parameter-sweep applications into jobs that can be scheduled to be executed on



Input Executable Cell: $E$29

Executable File Mappings

Logical / Frien...	File Name	Full Path
calc	calc.exe	F:\ExcelGrid\bin\calc.exe
test	SampleExce...	F:\ExcelGrid\bin\SampleExcelGridAp...

Select Input Cell Range: $A$2:$B$11

Select Output Cell Range: $C$2:c11

Input Type: File [✓ Parameter]

Create New Job For Each: [✓ Row] Column

Output Files

Output File name

Figure 2: ExcelGrid Runner GUI.

resources (computer systems), managing them and collecting the results of the execution when finished. The Gridbus broker has the capability to locate and retrieve data from multiple data sources, select the best resources based on users' QoS requirements, and also redirect output to appropriate storage locations. The main features of the Gridbus broker [18] include support for parameter-sweep model applications, simple, extensible, middleware-independent architecture, service – oriented, data-aware and economy capability, and platform – independence (being based on Java it can be a standalone application, or embedded within a Web application).

III. Architecture

ExcelGrid is a user-level middleware system designed to harness enterprise and global grids developed using Alchemi and Gridbus broker which provide services to execute jobs on resources that are grid-enables using various low-level middleware technologies such as Globus, UNICORE and Alchemi. ExcelGrid is designed to distribute compute intensive jobs for execution on grids with the aim of achieving a performance boost in job execution, using the parameter-sweep model [2]. Multiple instances of a job are run in parallel on different grid nodes, with different parameters.

The architecture of ExcelGrid is shown in Figure 1. The main components of the ExcelGrid are the ExcelGridAddIn, ExcelGrid Runner, and the GridClient.

A. ExcelGrid Components

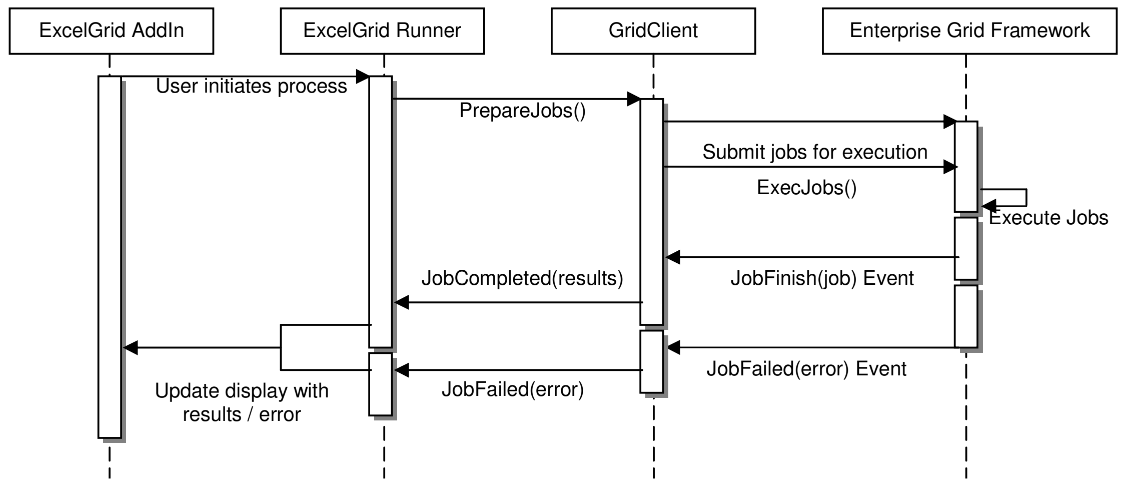
The ExcelGrid Add-In forms the interface to Excel. It provides access to the Excel application model. The AddIn component appears as a menu option in the standard Excel "Tools" menu. It is the connection between Excel and the ExcelGrid Runner component. The ExcelGrid Runner forms the main GUI (graphical user interface) as shown in Figure 2.. This is the main interface to the user from which required options are selected and job execution is initiated. It performs the functions of preparing the jobs to be submitted to the grid framework, based on the parameters input by the user. The runner also provides feedback to the user about the status of job execution on the grid.

A job is a work unit which performs any computation, and has a set of inputs and outputs. The input parameters for the job are extracted from the spreadsheet. The GridClient component attempts to connect to the grid middleware and submit the jobs for execution. The data in the spreadsheet is converted into appropriate job description formats for the grid, and jobs are requested to be scheduled. The GridClient can communicate with both enterprise and global grid frameworks. For enterprise grids, the Job API provided by Alchemi is used. The user selections are mapped to the job model of Alchemi to prepare and schedule jobs on worker nodes. To interact with global grids, GridClient communicates with the Gridbus broker. The communication is using raw socket communication and a protocol which describes the jobs and its parameters. All the job parameters are passed as a single message which the server splits to create the jobs. The Gridbus broker is used as a standalone server application in this case, listening for connections and providing brokering services for global grids. The GridClient listens for return messages from this server, and updates job status of scheduled jobs.

IV. Design and Implementation

Based on the architecture described in the previous section, we decided to implement the ExcelGrid components using object-orient design and development methodologies, using UML tools and .NET languages like VB.NET and C#. The interface to the Gridbus broker was implemented in Java. Figure 3 shows the interaction diagrams between the GridClient (AlchemiClient / GridbusClient) and the grid frameworks.

The basic process involved the same steps for both cases. The user initiates the execution, and the GridClient prepares the jobs to be submitted to the framework. In case of Alchemi, the jobs are submitted via the GJob API of Alchemi and call-back events/functions are registered with the application object. When the "JobFinished" or "JobFailed" events occur, the information obtained is passed on to the ExcelGrid Runner which updates the display in Excel and the progress form. In case of the Gridbus broker the protocol is slightly different. Since the broker is implemented in Java, and the GridClient in C#.NET direct call-back communication is not possible. Hence a simple socket based communication is used. Jobs are submitted using a single long message which combines all the parameters into a string. This message is received by the server and split into separate jobs. These jobs are then scheduled on the grid. The GridClient maintains a connection to the server while the jobs are executed. The server polls the job status, and



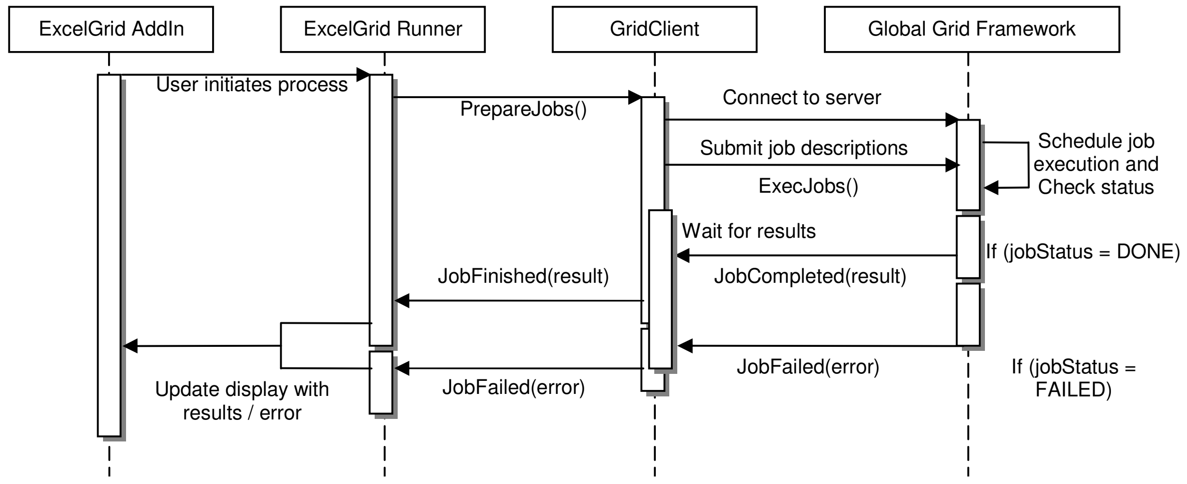


Figure 3: ExcelGrid Interaction diagrams.

reports completion / failure messages with results or error messages respectively. These messages then trigger events from the GridClient which are captured by the ExcelGridRunner to update the display.

Figure 4 shows the main classes in the ExcelGrid system. The ExcelGridAddIn component contains the “Connect” class which implements the Extensibility.IDTExtensibility2 interface. Its main function is the connection between Excel and the ExcelGridRunner component. By implementing the IDTExtensibility2 interface, the add-in can be loaded in Excel when it starts up, and forms an integral part of the Excel GUI. The ExcelGridAddIn contains the classes shown in Figure 5. The “Connect” class implements the IDTExtensibility2 interface, which involved implementing the methods – OnConnection, OnDisconnection, OnAddInsUpdate, OnStartupComplete, and OnBeginShutdown. These methods are call-backs called by Excel when a certain event happens in the GUI.

Microsoft Excel exposes an object model, which can be used to programmatically control it. The object model is a collection of classes and methods that serve as counterparts to the logical components of Excel. For example, there is an Application object, a Workbook object, and a Worksheet object, each of which contain the functionality of those components of Excel. This process of controlling Excel programmatically is called Automation [9][10]. Using automation, various actions such as invoking Excel, creating workbooks, adding and manipulating data in a worksheet etc can be performed. Excel automation is based on Microsoft’s COM (component object model) technology. COM and .NET cannot natively communicate with each other. However, a .NET feature called COM interop provides callable wrappers to allow .NET and COM to interoperate. A runtime callable wrapper (RCW) allows a

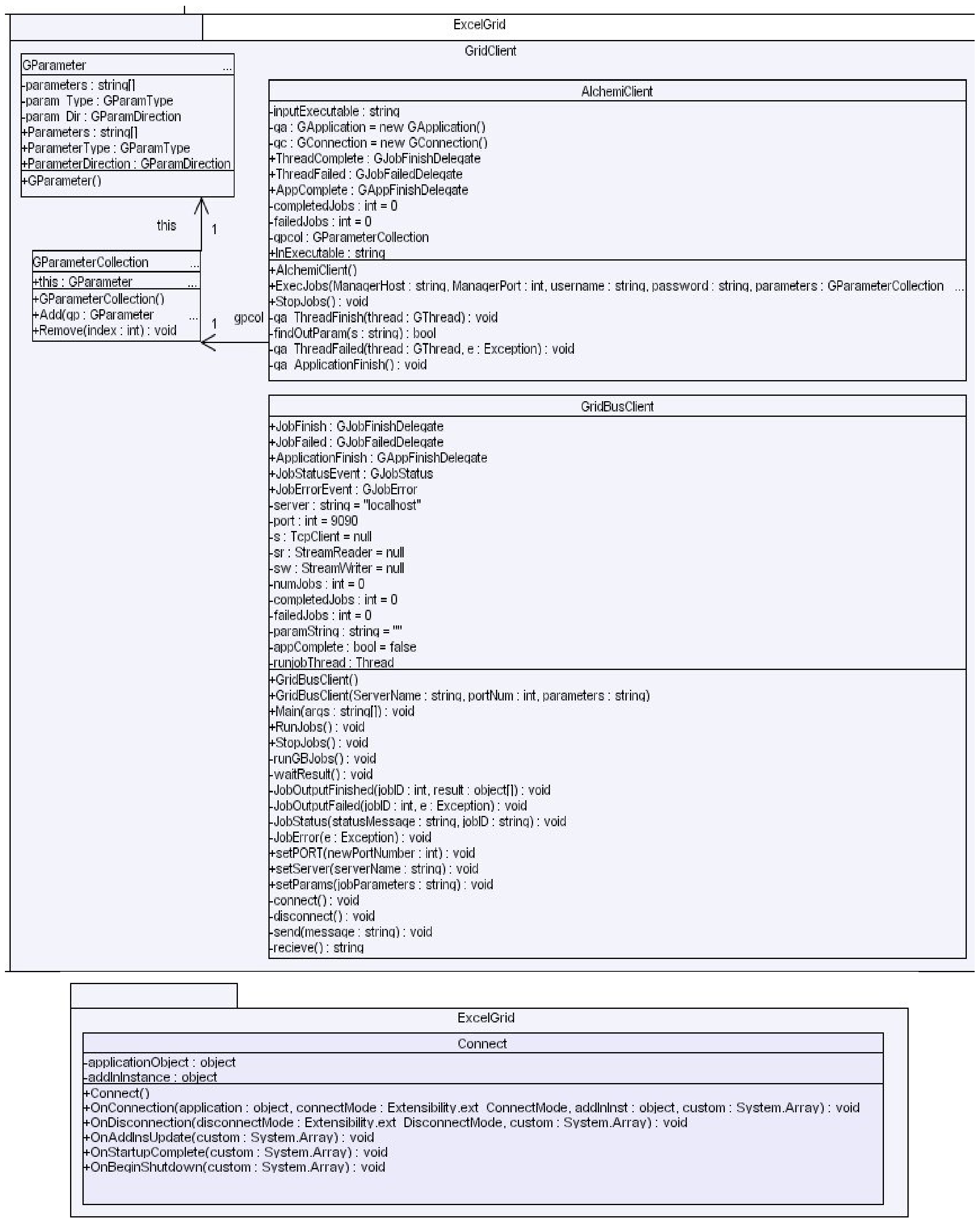

ExcelGrid
GridClient
GParameter
-parameters : string[]
-param Type : GParamType
-param Dir : GParamDirection
+Parameters : string[]
+ParameterType : GParamType
+ParameterDirection : GParamDirection
+GParameter()
this
1
GParameterCollection
+this : GParameter
+GParameterCollection()
+Add(gp : GParameter
+Remove(index : int) : void
1
gpcol
AlchemiClient
-inputExecutable : string
-ga : GApplication = new GApplication()
-gc : GConnection = new GConnection()
+ThreadComplete : GJobFinishDelegate
+ThreadFailed : GJobFailedDelegate
+AppComplete : GAppFinishDelegate
-completedJobs : int = 0
-failedJobs : int = 0
-gpcol : GParameterCollection
+InExecutable : string
+AlchemiClient()
+ExecJobs(ManagerHost : string, ManagerPort : int, username : string, password : string, parameters : GParameterCollection
+StopJobs() : void
-ga_ThreadFinish(thread : GThread) : void
-findOutParam(s : string) : bool
-ga_ThreadFailed(thread : GThread, e : Exception) : void
-ga_ApplicationFinish() : void
GridBusClient
+JobFinish : GJobFinishDelegate
+JobFailed : GJobFailedDelegate
+ApplicationFinish : GAppFinishDelegate
+JobStatusEvent : GJobStatus
+JobErrorEvent : GJobError
-server : string = "localhost"
-port : int = 9090
-s : TcpClient = null
-sr : StreamReader = null
-sw : StreamWriter = null
-numJobs : int = 0
-completedJobs : int = 0
-failedJobs : int = 0
-paramString : string = ""
-appComplete : bool = false
-runjobThread : Thread
+GridBusClient()
+GridBusClient(ServerName : string, portNum : int, parameters : string)
+Main(args : string[]) : void
+RunJobs() : void
+StopJobs() : void
-runGBJobs() : void
-waitResult() : void
-JobOutputFinished(jobID : int, result : object[]) : void
-JobOutputFailed(jobID : int, e : Exception) : void
-JobStatus(statusMessage : string, jobID : string) : void
-JobError(e : Exception) : void
+setPORT(newPortNumber : int) : void
+setServer(serverName : string) : void
+setParams(jobParameters : string) : void
-connect() : void
-disconnect() : void
-send(message : string) : void
-recieve() : string
ExcelGrid
Connect
-applicationObject : object
-addInInstance : object
+Connect()
+OnConnection(application : object, connectMode : Extensibility.ext_ConnectMode, addInInst : object, custom : System.Array) : void
+OnDisconnection(disconnectMode : Extensibility.ext_DisconnectMode, custom : System.Array) : void
+OnAddInsUpdate(custom : System.Array) : void
+OnStartupComplete(custom : System.Array) : void
+OnBeginShutdown(custom : System.Array) : void

Figure 4: ExcelGrid UML diagrams.

COM component (for example, an Office VBA object library) to be used by .NET (for example, a Visual Basic .NET application). After the .NET assemblies are generated, classes and their members can be instantiated and invoked from Visual Studio .NET as if the COM objects and members were native .NET classes and members. The ExcelGridAddIn is a .NET component which interacts with Excel's object model using COM interop [13].

The ExcelGridAddIn invokes the ExcelGridRunner component which is a standard Windows GUI application The ExcelGridRunner obtains the currently running instance of Excel, and communicates with it, via automation. The ExcelGridRunner is the component which detects the user's actions in Excel, and takes input from the currently active Excel sheet, and creates and runs jobs on the grid, by invoking the AlchemiClient / GridbusClient classes in the "GridClient" component. It contains the frmGClient class which is the GUI component of ExcelGridRunner.

GridbusClient / AlchemiClient (depending on the chosen grid framework), in a GParamCollection object.

When the user initiates the execution process from the GUI, input from the selected rows and columns is converted into GParameter type objects, before being submitted as jobs to the grid client. The GridClient (AlchemiClient / GridbusClient) performs the actual communication with the grid framework.

The GridbusClient / AlchemiClient object passes messages to the main grid node, for each job that is to be run on the grid, and listens to events / messages from that node. On receipt of a message / event this object then raises events to signal

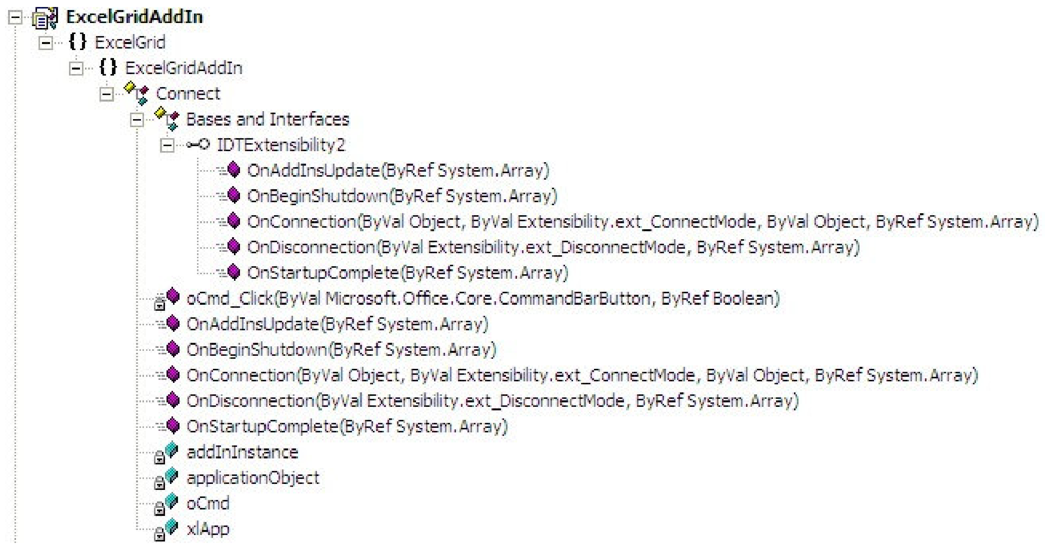

ExcelGridAddIn
ExcelGrid
ExcelGridAddIn
Connect
Bases and Interfaces
IDTExtensibility2
OnAddInsUpdate(ByRef System.Array)
OnBeginShutdown(ByRef System.Array)
OnConnection(ByVal Object, ByVal Extensibility.ext_ConnectMode, ByVal Object, ByRef System.Array)
OnDisconnection(ByVal Extensibility.ext_DisconnectMode, ByRef System.Array)
OnStartupComplete(ByRef System.Array)
oCmd_Click(ByVal Microsoft.Office.Core.CommandBarButton, ByRef Boolean)
OnAddInsUpdate(ByRef System.Array)
OnBeginShutdown(ByRef System.Array)
OnConnection(ByVal Object, ByVal Extensibility.ext_ConnectMode, ByVal Object, ByRef System.Array)
OnDisconnection(ByVal Extensibility.ext_DisconnectMode, ByRef System.Array)
OnStartupComplete(ByRef System.Array)
addInInstance
applicationObject
oCmd
xlApp

Figure 5: ExcelGridAddIn class hierarchy

The class hierarchy of the ExcelGridRunner component is shown in Figure 6. The ExcelGridRunner is built as a standard Windows Application from the standard project template for Windows Forms Application in Visual Studio.NET. It consists of the frmGClient and frmProgress classes which form the GUI components the user interacts with. The GSetting class maintains stores user's settings like server details, job executables and input / output options selected in Excel and stores them to disk so that they are available across different runs / invocations of the program. The GSetting class uses the Mappings class to store the mapping of file locations and logical names of the job executables. The mainClass initiates the execution of the ExcelGridRunner component.

The computation that needs to be done on the grid nodes is represented as work units known as jobs. Each job has an associated executable which is the program that executes on the remote node. The job also has input and output parameters which are represented by the GParameter class. A collection of GParameter objects is passed to an object of type

completion / failure of threads and completion of the application running on the grid, which are handled by the frmGClient, which obtains the output and inserts it into Excel using OLE automation.

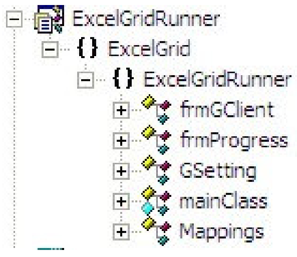


Figure 6: ExcelGridRunner Class Hierarchy.

Using the GridClient component, the executable job, as well as the input parameters (which may include files) are all packed and sent to the remote node. The results and output files are received from the remote node. All this process is automatically taken care of by the grid frameworks – Alchemi / Gridbus broker.

From the ExcelGrid GUI the user can select which ranges of cells contain the parameters that are used for each job on the grid. The user can decide if each row should generate parameters for a new job, or each column value should create a new job. The user can also specify where the output needs to be placed. The user should specify what executable contains the computations that need to be performed on the selected data. This executable can be given a friendly / logical name. The logical name in the Excel sheet is used by ExcelGrid to map to the actual executable on the disk.

V. Performance Evaluation

To evaluate the performance characteristics of job execution on a grid when invoked via the ExcelGrid, we used a synthetic test application that calculates mathematical functions based on the values of two input parameters. The first parameter X is an input to a mathematical function and the second parameter Y indicates the expected calculation complexity in minutes plus a random deviation value between 0 to 120 seconds—this creates different parametric jobs similar to a real application. The calculation complexity is the number of times the mathematical function is executed per task.

We have tested the performance of job execution on Excel using ExcelGrid on both Alchemi nodes and Gridbus nodes. The ExcelGrid application was run from a laptop located at the GRIDS laboratory at the University of Melbourne, Australia. This client communicated with the grid framework installed on the test bed machines. The test bed configurations and results for each case are outlined below.

A. *Enterprise Grids using Alchemi*

An Alchemi enterprise grid consisting of five worker nodes (Pentium IV desktop PCs) based in the GRIDS laboratory, at the CSSE Department, Melbourne University was used. One of the machines was designated as an Alchemi Manager node.

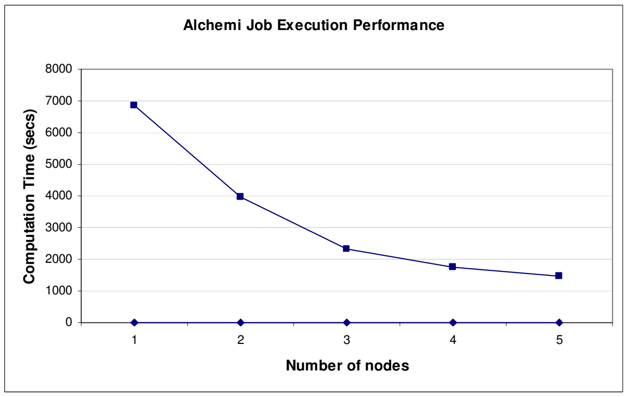


Figure 7: Enterprise Grid Performance variation with # of nodes

Executor Node	Configuration
bart.cs.mu.oz.au	Intel Pentium4 1.7Ghz CPU
lisa.cs.mu.oz.au	Intel Pentium4 1.7Ghz CPU
pc356.cs.mu.oz.au	Intel Pentium4 1.8Ghz CPU
rajpc3.cs.mu.oz.au	Intel Pentium4 2.4Ghz CPU
rajpc5.cs.mu.oz.au	Intel Pentium4 2.4Ghz CPU

Table 1: Alchemi Test bed resources.

The node configurations used are shown in Table 1. Using the application mentioned above five tests, with varying number of nodes was performed using the Enterprise grid. 50 jobs were run at the same complexity on a grid of 1,2,3,4, and 5 nodes. Each node was running the Alchemi Executor v0.8 in dedicated mode. The manager node was running from the machine named Bart. The graph in Figure 7 shows the improvement in performance with the increase in the number of nodes in the grid.

B. *Global Grids using Gridbus Broker and Globus Middleware*

For the Gridbus experiment the Belle analysis test bed data grid, which has resources distributed around Australia including Melbourne, Sydney and Canberra was used. These

Server Name	Owner Organisation	Configuration	Grid Middleware
belle.cs.mu.oz.au	GRIDS Lab, The University of Melbourne	IBM eServer with 4 CPUs.	Globus v.2.4
belle.anu.edu.au	Australian National University, Canberra	IBM eServer with 4 CPUs.	Globus v.2.4
belle.physics.usyd.edu.au	The University of Sydney, Sydney	IBM eServer with 4 CPUs.	Globus v.2.4
fleagle.ph.unimelb.edu.au	The University of Melbourne	Desktop PC with 1 CPU	Globus v.2.4
brecca-2.vpac.org	VPAC Melbourne	IBM 2 CPU SMP server	Globus v.2.4

Table 2: Test bed resources.

systems are interconnected via GrangeNet (Grid and Next generation Network) which is a multi-gigabit network supporting grid and advanced communication services across Australia. The test bed resources are shown in Table 2.

The belle system at the University of Melbourne was used to deploy the broker, and the agents were dispatched to other resources at runtime by the Gridbus broker.

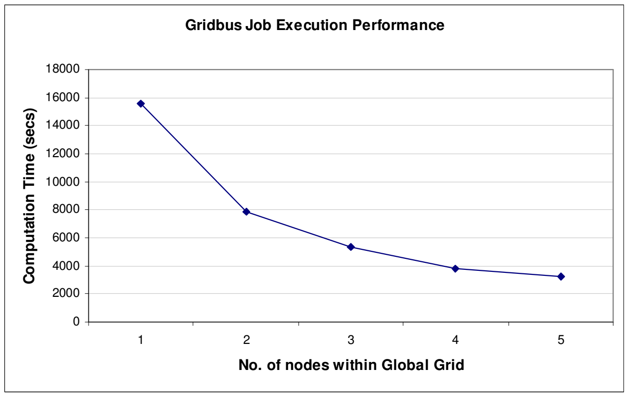


Figure 8: Global grid job execution performance.

The performance tests aimed to determine the effect of increasing number of grid nodes for a fixed job size and number of jobs. The sets of resources used for the tests are shown in Table 3. The findings of the performance evaluation experiments are shown in Figure 8.

The graph shows the job execution performance of the grid nodes when managed by the Gridbus broker. The computation time versus number of nodes used in the test, is plotted, and we see that as the number of nodes increases the time taken to execute the jobs reduces, but the speed up is not linear.

Configuration	Resources
1	belle.cs.mu.oz.au
2	fleagle.ph.unimelb.edu.au, brecca-2.vpac.org
3	fleagle.ph.unimelb.edu.au, brecca-2.vpac.org, belle.physics.usyd.edu.au
4	fleagle.ph.unimelb.edu.au, brecca-2.vpac.org, belle.physics.usyd.edu.au, belle.anu.edu.au
5	fleagle.ph.unimelb.edu.au, brecca-2.vpac.org, belle.physics.usyd.edu.au, belle.anu.edu.au, belle.cs.mu.oz.au

Table 3: Test bed resource configurations.

VI. Related Work

There has been a good amount of work going on in the area of developing spreadsheet based applications and front-ends to grid systems. Some of the work in this area is summarised in Table 4.

The InnerGrid Nitya system uses a proprietary API and requires installation of the Excel connector (which is an Excel Addin) on each computer, to use the features which are available through the Excel "Tools" menu. It works on the Windows platform (clients and servers). The InnerGrid Excel connector allows the use of grid technology by distributing costly spreadsheet calculations among corporate PCs connected via an intranet, reducing the execution time and thus scaling the computational capabilities [8].

The Platform Symphony Adapter for Microsoft Excel software accelerates online and batch operations for business critical financial services applications. This new feature greatly enhances the processing speed of compute-intensive Microsoft Excel spreadsheets by transparently distributing the calculations to an allocated number of clusters. The result is lower calculation times, an increase in server utilisation and guaranteed trader service levels which in turn induces higher trading revenues, profits and lower business risk. The Adapter does not require any modifications to Excel, and is implemented via XLL compiled add-ins. The Adapter works on the Windows platform (clients and servers) [14].

ActiveSheets builds on a research tool called Nimrod [2], which is a specialised parametric modelling system, by adding a component based spreadsheet interface for specifying computational experiments. These tools can be used to distribute independent executions of a single simulation in order to perform complex scenarios analysis. They manage the generation of work, distribute the computation, and gather the results. The output is often read back into a single spreadsheet for analysis and visualisation after it has been returned from the distributed computers. The mechanism for the parallel evaluation of spreadsheets is based on the dataflow model of computation [3][5].A custom function sends its arguments together with a representation of the function to backend computer for parallel evaluation [4] A table is used to store the current state of the cell, which can be one of the following states: unevaluated, under evaluation or evaluated. The ActiveSheets system is described in detail in [1].

VII. Conclusion

Grid computing has proven to be extremely advantageous to the executions of spreadsheets. Successful deployment of ExcelGrid will be beneficial to data/compute intensive grid computing such as drug designs and weather forecasting. ExcelGrid is designed to provide extensions to Excel to add the ability to execute custom and legacy jobs on a grid, without requiring rewriting the applications for the grid. The familiar GUI of Excel and the easy-to-use interface of ExcelGrid itself, which is invoked via Excel, can make it a very powerful and useful tool.

Acknowledgements

The work presented in this paper is partially supported through the Australian Research Council Discovery Project Grant and

Grid Plugins	Description	Remarks
ActiveSheets	A research mechanism for parallelising traditional spreadsheets, ActiveSheets is a component based spreadsheet interface for specifying computational experiments. Implements a two stage evaluation process for custom functions.	ActiveSheets works with Nimrod and Netsolve grid middleware systems and provides parallelism at the spreadsheet function level. Custom functions are used to encode the parallel evaluation of multiple cells. ActiveSheets is not open-source software.
InnerGrid Nitya	A technology from GridSystems company, that grid-enables Excel, by distributing spreadsheet calculations among corporate PCs, mainly targeted at financial markets.	This is commercial software and is not open-source. InnerGrid Nitya works on enterprise grid systems.
Platform Symphony	Platform introduced the Adapter for Microsoft Excel which aims to accelerates online and batch operations for business critical financial services applications, by transparently distributing the calculations to an allocated number of clusters.	This is commercial software and is not open-source. Platform Symphony is designed to operate on enterprise grids.
ExcelGrid	ExcelGrid is a grid front-end to Excel, developed at the University of Melbourne, which enables users to run custom jobs, on any grid system – Enterprise level grids or Global grids.	ExcelGrid is open-source software that works with both enterprise (Alchemi) and global grid frameworks (Gridbus broker) and provides a job submission interface, which is parallelism at the job-level. Custom and legacy jobs are grid enabled to achieve parallel execution.

Table 4: Representative works in Spreadsheet processing on parallel/distributed systems.

Storage Technology Corporation sponsorship of Grid fellowship. We would like to thank Wendy Cameron for her original implementation of a Java-based server working as an interface to the Gridbus broker. We have extended this server to create an interface between ExcelGrid and the Gridbus broker.

REFERENCES

[1] D. Abramson, P. Roe, L. Kotler and D. Mather, “ActiveSheets: Super-Computing with Spreadsheets”, 2001 High Performance Computing Symposium (HPC'01), part of the Advanced Simulation Technologies Conference, April 22-26, Seattle, Washington, USA.

[2] D. Abramson, R. Sosic, J. Giddy and B. Hall, “Nimrod: A Tool for Performing Parametised Simulations using Distributed Workstations”, The 4th IEEE Symposium on High Performance Distributed Computing, Virginia, August 1995.

[3] L. Arvind and T. Ungerer, “Evolution of Dataflow Computers”, Chapter 1, “Advanced Topics in Dataflow Computing”, Prentice Hall, 1991.

[4] G. Cheuk, ActiveSheets: Grid Computing with Spreadsheets, The First Australian Grid Forum Workshop (OzGrid-1) Dec. 9-10, 2002, Melbourne, Victoria, Australia.

[5] J. Dennis, “The Evolution of ‘Static’ Data-flow Architectures”, Chapter 2, “Advanced Topics in Dataflow Computing”, Prentice Hall, 1991.

[6] Excel for Business Intelligence, September 9, 2003. http://www.microsoft.com/office/previous/xp/business/intelligence/excel.asp

[7] I. Foster and C. Kesselman, Globus: A Metacomputing Infrastructure Toolkit, International Journal of Supercomputer Applications, 11(2):115-128, 1997.

[8] GridSystems, InnerGrid Nitya http://www.gridsystems.com/pdf/IGFinancials01.pdf , May 11, 2004.

[9] How to automate Microsoft Excel from Visual Basic .NET, Microsoft Knowledge Base,http://support.microsoft.com/default.aspx?scid=http://support.microsoft.com:80/support/kb/articles/Q301/9/82.ASP&NoWebContent=1

[10] How to automate Microsoft Excel from Visual Basic, Microsoft Knowledge Base, http://support.microsoft.com/default.aspx?kbid=219151

[11] J. Almond and D. Snelling, UNICORE: Uniform Access to Supercomputing as an Element of Electronic Commerce. Future Generation Computer Systems 613(1999), 1-10.

[12] A. Luther, R. Buyya, R. Ranjan and S. Venugopal, “Alchemi: A .NET-based Grid Computing Framework and its Integration into Global Grids”, Technical Report, GRIDS-TR-2003-8, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, December 2003.

[13] P. Cornell, Creating Office Managed COM Add-Ins with Visual Studio .NET, Microsoft Corporation June 6, 2002, http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnofftalk/html/office06062002.asp

[14] Platform Symphony, Platform Computing Press Releases, http://www.platform.com/newsevents/pressreleases/prelease.asp?id=82, May 6, 2004.

[15] D.J. Power, "A Brief History of Spreadsheets", DSSResources.COM, World Wide Web, http://dssresources.com/history/sshistory.html, version 3.5, 10/04/2003. Photo added September 24, 2002.

[16] Private Grid, Netherlands, http://www.private-grid.nl/

[17] I. Foster and C. Kesselman (eds.). *The Grid: Blueprint for a Future Computing Infrastructure*. Morgan Kaufmann: San Francisco, CA, 1999.

[18] S. Venugopal, R. Buyya and L. Winton, A Grid Service Broker for Scheduling Distributed Data-Oriented Applications on Global Grids, Technical Report, GRIDS-TR-2004-1, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, February 2004.

[19] J-Walk & Associates, The Spreadsheet Portal, Excel Section, http://www.j-walk.com/ss/excel/, May 2004.

[20] A. Oram (ed.), *Peer-to-Peer: Harnessing the Power of Disruptive Technologies*. O’Reilly Press, U.S.A., 2001.