HETEROGENEOUS COMPUTATION GRID

(a peer-to-peer node-based resource virtualization system)

Arijit Sengupta

and

Asankhaya Sharma

B.Tech (Computer Science and Engineering),
National Institute of Technology, Warangal

C/o Department of Computer Science and Engineering,
National Institute of Technology Warangal,
Warangal, Andhra Pradesh – 506 004.
Email: arijitseng@yahoo.com, asankhaya@yahoo.com

HETEROGENEOUS COMPUTATION GRID

(a peer-to-peer node-based resource virtualization system)

ABSTRACT

In this paper, we present *Heterogeneous Computation Grid* (HCG), a peer-to-peer node-based resource virtualization system. HCG is designed to provide a decentralized architecture for resource sharing in solving computationally intensive problem by use of idle time of client devices across multiple platforms. HCG also enables these peers to request individual services from the grid. Traditional grid computation requires a central grid server to dispatch work and get progress update from clients. The grid is used for a limited set of server-defined projects, often with no benefits to the client. The HCG model has no server-defined projects, with the resources of the entire grid at the disposal of peers. It is evident that creating a sustainable P2P model for resource virtualization is a mammoth task. HCG accomplishes these goals by maintaining a database of projects in the grid and a list of registered users. The client program on the peer device schedules idle time for the execution of jobs in the database, subject to choice and priority constraints. The client also doubles up as a job management unit, which submits jobs to the grid and collates the results, while securing the data and the internal working of the software from the peer. HCG also derives its computational power from numerous Applets and ActiveX controls that are distributed on websites. It can also use the computational power of mobiles and embedded devices that can connect to the grid. HCG will be most useful for academic institutions and corporate houses, which can pool up their resources for the most computationally intensive work, without any loss of confidentiality or data security.

Keywords: grid computing; P2P; embedded devices; Java; heterogeneous computation

INTRODUCTION

The rapid proliferation of the Internet and the rise in computational power of machines has provided us the potential for harnessing vast numbers of computers, storage devices, and networks as a platform for computation. The computational power of a vast number of computers, in academic institutions, corporate houses and even homes lie unused due to the poor frameworks for grid resource utilization and absence of industry-standard interfaces [1]. The main challenges of such an interface would be to provide a secure, reliable, efficient and economically viable [2] model for grid computation. For distributed programming to make headway in the day-to-day computation, it is imperative that the grid model changes from a server-client model to a peer-to-peer (P2P) network-based model. The proposed model uses a P2P network to create a computational platform encompassing systems of varied architecture and capabilities. Any node in the network can effectively utilize the idle computational time and power of peer nodes and donate its own idle time for the computations of other nodes.

Two major design goals of this project include peer-based modelling and heterogeneous peers with cross-platform compatibility and architecture-independent implementation. Important questions of scalability, reliability, security and confidentiality also needed to be addressed.

Heterogeneous Computation Grid (HCG) is a large interconnection of smaller networks. Analogous to the system of Local Area Networks (LAN) and the Internet, there can be several HCG implementations at different levels. The internet-analogous HCG network will consist of a central repository which holds a record of registered global projects and participating users. In addition, smaller HCG networks will also exist in parallel and will cater to local requirements. Such HCG networks may dispense with the central server altogether in favour of a Distributed Hash Table (DHT) lookup or workgroup-share listing for enumeration of users and projects. Both these networks consist of several nodes which will communicate using UDP on fixed unassigned port numbers for smaller networks and XML data transfer on port 80 (HTTP) for larger networks. Such nodes can double up as program solvers (module called client) or problem posers (module called manager). The manager module consists of a Dispatcher, for dispatch of program and problem segment (or URI to the same) and an Acceptor, for collation of received data. The client module is responsible only for scheduling external projects during the idle time of the machine. Computationally less-intensive projects may be delegated to Java Applets and ActiveX controls which can perform the client machine with user permission, communicate with the server and return results to the corresponding manager's acceptor module. The suggested framework also allows for the usage of heterogeneous systems as diverse as any embedded device which has the capability to connect to the internet. The wide degree of heterogeneity offered by the model and the absence of requirement of any server is expected to accelerate the pace of distributed computations.

RELATED WORK

Distributed computing was born in 1970 when computers were first linked by networks. The first major project was *Creeper and Reaper*, which ran on the Arpanet. In 1970, Shoch and Hupp created a "worm" that moved from machine to machine, utilizing idle CPU cycles. However, these models soon became obsolete with the maturation of the Internet in the 1990s. The first global project, called distributed.net (also called *dnet*) [3], used thousands of independently owned computer systems to crack encryption codes, taking distributed computation to a new level. The second, and the most successful and popular of distributed computing projects in history, is the *SETI@home* project [4]. Over two million people – the largest number of volunteers for any Internet distributed computing project to date – have installed the SETI@home software agent since the project started in May 1999. This project conclusively proved that distributed computing could accelerate computing project results while managing project costs, thus firmly establishing grid computation as is known today. The Berkeley Open Infrastructure for Network Computing (BOINC) [5] has been made use of for this project. While the BOINC defines most major communication protocols for use in distributed computation and is close to a de-facto standard for grid computation, most contemporary implementations use a model that make either of the following two implicit assumptions:

1. The project is a large-scale project, with resources available to scale. For small scale projects, the server-client model in use by all current implementations is superfluous and can be replaced easily by an equal-configuration model wherein no special and specific system requirements may be made of a dedicated server. Models such as Beowulf [6] and Globus [7, 8, 9] limits the feasibility of the model in all but few very large projects, where it is viable to set up dedicated server with large storage capacities, processing power and network bandwidth.

2. The system architectures of the grid nodes are limited by problem specifications. Most current implementations require nodes to have processing power of at least 266MHz. Dependence of the program to be executed (or the implementation itself) on system or network architecture is also a major limiting factor. Making the use of platform-independent languages, such as Java, compulsory for the implementation is not a solution, as Java™ [10] byte-codes (also any other interpreted language) run slower than native code due to the presence of the additional layer of a virtual machine, and creates a trade-off between execution speed and program portability.

To resolve the first issue, the requirement of a server can be done away with the adoption of a peer-to-peer model. This will bring the grid computation out of the realm of extremely large-scale worldwide internet projects and provide a computing paradigm similar to an electric power grid – a variety of resources contribute power onto a shared pool for all clients to access as and when needed. Work on the peer-to-peer grid computational model has not made much headway, and the ideal grid computational scenario mentioned above is still in the distance. While a peer-to-peer model may solve several of the problems faced by current grid networks, and remove the necessity of dedicated grid infrastructure plans for customers in their own facilities; it also introduces several of its own problems. Most of these problems have been listed and solutions have been proposed for them in this paper.

The second problem can be solved for introducing more flexibility into the grid computational framework. Executables for several architectures and in different formats may be provided by the problem poser and the client program must decide which one to download and run based on a per-PC choice. Different systems may also be given different work. Higher-end servers may be given more computationally intensive work while lower-end embedded devices may be delegated work of verifying non-critical results. Work in this area is progressing rapidly, but the essential solution is a trade-off between support for more architectures and systems, and porting of programs.

PROPOSED ARCHITECTURE

Basic Design

The basic grid design must support the following two operations:

1. Every PC on the network must be known, so as to provide its idle time to the network.
2. The problem to be distributed must exist on a computer (or set of computers) on the network.

There are therefore two types of systems on the network in the proposed model – Problem Poser and Problem Solver (hereafter called client). Getting a list of problems and list of clients is a tricky issue to be considered here and is solved in the following ways.

- On a local area network[1] (LAN) configured as a domain, the list of clients can be the list of systems can be obtained as the list of computers logged on to the server. If the LAN is configured as a workgroup, the list of systems is again available. This is however, not true of wide area network (WAN) and metropolitan area networks (MAN). The list of projects can be implemented as shares on each PC on the network.

- On large-scale networks such as WANs and the internet, two approaches may be used. A server may be used to house a database listing of users currently on the project and the number of active projects. This is a feasible solution if the number of computers is limited to a few thousands, but becomes unviable if project and user data require the database to be updated frequently or the size of the repository maintained becomes too large. In such a scenario, a peer-to-peer (P2P) architecture can be used to maintain the database shared among users. Distributed Hash Table (DHT) lookup can be used to find out active users and check for current projects. Such a model is illustrated in Figure 1. It is important to note that the HCG model does not preclude a server; it only provides an alternative if a dedicated server of capacity is unavailable. The model therefore works as well with a server, which allows effective monitoring of the processes currently in progress (refer [11, 12, 13] for the server model).

The extension of HCG onto networks other than LAN and internet is also possible. However, as of this moment, no trails have been made on the extensibility of the proposed architecture to other networks. There is no experimental data available on the adaptation of HCG to other networks.

[1] The idea of local area network, as mentioned in this document, is a subset of the concept of an Intranet, and can be extended to any network on which shared resources (in this case, projects) are enumerable. This is irrespective of the type of network connectivity used: Ethernet, wireless and so on. The term Intranet has been avoided due to its extensive domain.

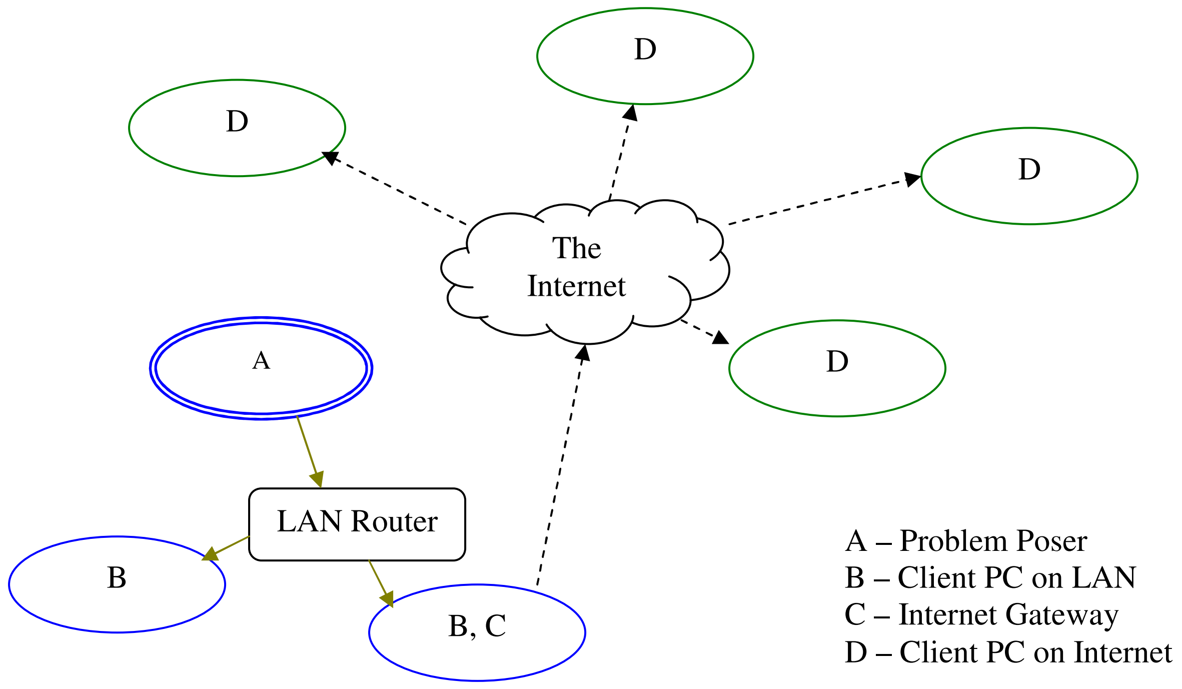


Figure 1. General Model of a server-less P2P HCG

Server Side

The server (when present) maintains a database of the following data:

- List of participating (or registered) and logged-on users.
- List of current projects
- List of tasks for each project.
- List of associations of users and tasks and task-status.

All this data may be maintained as tables on the server side. It is important to note that while HCG defines the data that can be stored on the server and the fields of the tables used, it does not mandate the presence of any of these tables. The data stored on the server must conform to the HCG standard in that it must contain the stipulated fields of the tables mentioned, if it does contain the table, but may also include extra fields and constraints in the table. A brief overview of the database schema to be used is included in Appendix A.

Client Side

The client is used here to refer to any system that does not act as a server. The client system supports both submission and execution of projects. The submission part is handled by a module called the Manager, and the execution of projects take place in a component that will hereafter be referred to as the Client.

Manager

The manager submits programs to the grid for computation. In any resource virtualization, there must be checks and balances to prevent hogging of the grid. This is effected in HCG by a priority system. Each user of a manager is assigned priority, which can be pre-assigned by the grid administrator, or set to a default value in the absence of grid administrative rules. Priority decreases when a user's program takes its share of the grid, and increases when the user makes his system available for computation on the grid. Programs are assigned priority by their submitter which may be no more than the priority of the schedules. Programs are distributed in proportion to their priorities. Any user, however reserves the right to change priorities of programs to be executed on his system. Special values of priority (above 9 on a scale of 10) are provided to pre-empt execution of current programs on the grid and use the computational power of the grid for that computation solely. This feature can be made use of for mission-critical or real-time computations.

The Manager has two major components – the dispatcher and the acceptor. The HCG specification does not require both these programs to exist on the same system. However, this is generally the preferred option on most systems for small-scale programs. As problem complexity rises, more than one dispatcher may be mentioned and similar is the case for acceptors. The maximum number of dispatchers and acceptors is 249 each. When multiple dispatchers are given, one may be chosen at random.

Dispatcher

The dispatcher(s) takes an input string containing machine type and returns the address to the most suitable resource to be transmitted. The resource may then be downloaded onto the target PC. The dispatcher is designed as a web server for internet and as a port listener for a pre-assigned port on LAN. Communications on LAN are generally using UDP, with call-back to inform message receipt for critical messages, or on demand. These messages may be encrypted using DES (with MD5 hashing) if required to maintain security. Symmetric encryption is not supported for messages. Common messages are pre-encoded and the message structure is extensible to include future message families. If there is a server that maintains project information, the dispatcher is expected to keep the server updated with project status.

The dispatcher may maintain a list of tasks[1] for a project (refer BOINC [5]). The resource-set for each task is dispatched on demand. Each task has two reference counts – *DispatchCount* and *ResultCount*. DispatchCount is incremented each time a task is dispatched. ResultCount is incremented only when a result becomes available. Tasks are dispatched in ascending order of DispatchCount. When ResultCount increases beyond 1 (i.e. more than one result become available), the two results are compared. If equivalent, the DispatchCount variable is doubled; if not, the DispatchCount variable is decremented by ResultCount and ResultCount is decremented by 2. This model is used to protect against malicious intent and computational errors.

Acceptor

The acceptor(s) gets the computational output in text format (in methods very similar to dispatcher messages). Each message contains a message mode, and the message is limited to 251 bytes, or the limit of the messaging medium, whichever is lesser. For larger output messages, files may be sent over the network, or a uniform resource locator may be sent pointing to the location of the file. There is also a query message which requires path and privileges of user to upload output files on the internet. The acceptor is expected to reply to every message received, even if it is only a message receipt confirmation.

[1] Task is defined as that part of the project that can execute independently. Every project to be solved in HCG must consist of several tasks, which execute concurrently on separate PCs. Tasks may be dynamically created, removed and evaluated, but cannot be changed once created. Tasks do not have priorities, but may be assigned by the dispatcher. Each task involves one resource set to be downloaded to the system.

Client Program

The client program schedules the grid work on the user's system. On the client, the user has the choice of project to work on. Task scheduling is weighted by project priority, and the user need not mention his choices. In such a case, the choice of project to be worked upon is chosen on the weight of project priority.

Since unknown code executes on a client machine, there is a need to maintain security on the user's machine [14]. Technologies such as sandboxing and code signing [15] are supported for diverse platforms, and programs are expected to communicate with their manager (if required) only through the client program. By default, HCG does not provide scratch space [16, 17] to programs running in the sandbox. The format of executable programs supported may vary with the client and it is expected to provide a list of preferences to the manager, depending on operating system [18], when requesting a file.

Device Hierarchy

The heterogeneity of HCG is in its architecture which virtually places no bar on the type of device allowed to be used, except that the device is connectible to the network using either of the protocols defined. An illustration of the devices that are supported by HCG is given in Figure 2.

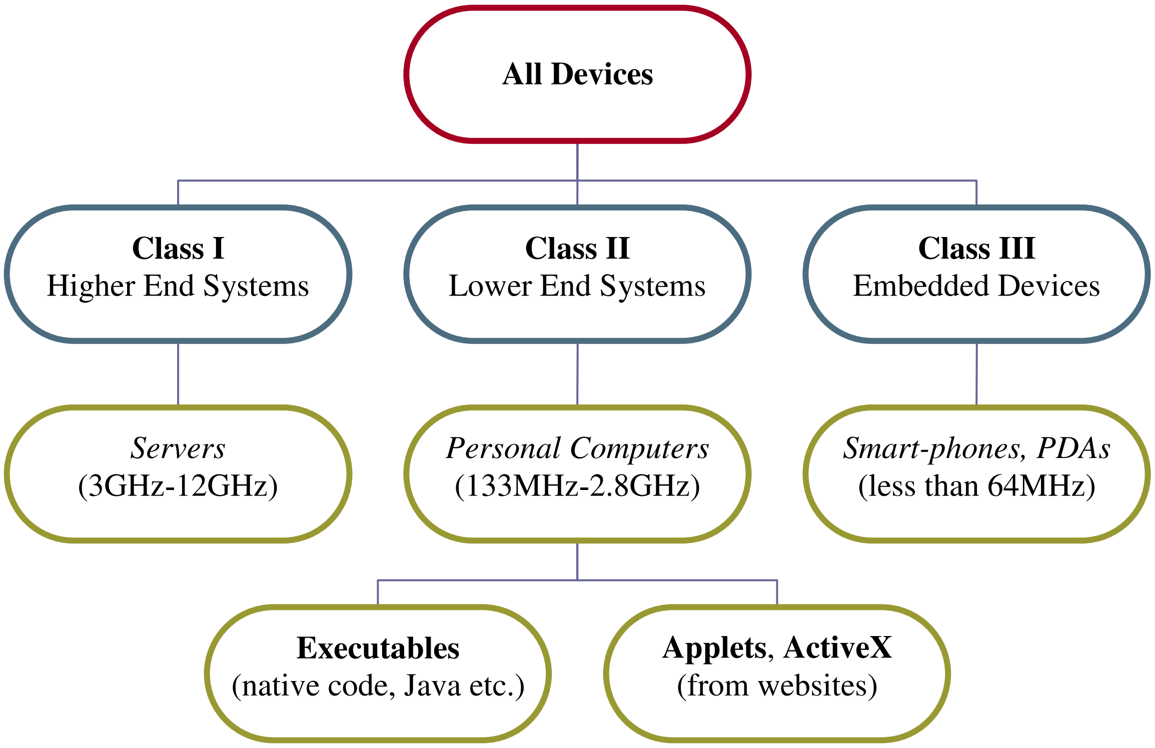


Figure 2. Devices supported by HCG

Use of Applets and ActiveX controls

The number of people who have access to broadband internet connection is steadily on the rise. A large section of people visit websites which offer e-books, novels, tutorials and other large sets of text. The user, in such a case visits a page for a long time. This time can be utilized (with the user's consent) for the computations of the grid. Applets and ActiveX controls embedded into these pages can download to the user's PC, initialize and run on the user's machine, giving progress indications to the user. These will generally be fast jobs, taking not more than few tens of minutes. The outputs can then be sent back to the network. While not a part of the design specifications of HCG, this is an integral part of the internet HCG. The applet project is hard-coded (or given as a parameter) but the task is not – the applet must get the task-set from the dispatcher. The task may also be given as an applet parameter by a server-side script. Java applets have the additional property of platform independence and can be used to implement web-based client/scheduler.

Use of Embedded Devices

The popularity of embedded devices[1] is increasing, despite the limited capabilities and computational power of these devices. Devices such as the mobile phone are kept switched on throughout the day. It is a known fact that which a mobile phone has very poor computational power, it uses merely a fraction of its capabilities. This is because the requirements of embedded devices peek at times and remain stable at low values at other times. Such a device is a good candidate for inclusion into the grid. Since embedded devices generally have poor computational capabilities, it is a recurring question on their contribution to the grid. Given below are some inherent advantages of embedded devices:

1. Programs written in Java™ can be ported to the mobile platform[2] [19].

2. Most embedded devices such as smart-phones need to (or are used to) communicate with the outside world, and thus have strong communication links. These communication links can be effectively utilized by the grid to communicate with the devices. The market is expected to drive the requirement for faster and higher-bandwidth connections.

3. The processing power of these devices is increasing rapidly, and poor computational power of these devices is all set to become a thing of the past. It can also be argued that the low power of these devices is offset by faster communication links.

HCG allows connection of embedded devices to the grid in three ways:

1. Through Bluetooth™ [20] technology (or data cable), mobile devices can communicate with the Manager of a computer. Problem specifications may be transferred through to the intermediate computer system. The system's HCG Manager adds a program entry and dispatches the program segments to the grid. During this period, the mobile device may be online or offline. Online refers to the condition wherein the mobile device receives constant intimation about the state of the computations and receives call-backs from the intermediate computer when computation results become available. The mobile device can then update the problem set or cancel the operation altogether. In offline mode, the intermediary collates the results and passes it on to the mobile on completion of work. Online mode can be used for real-time processing when connection speeds are significantly large, while offline mode will be

[1] The term embedded devices is used in this text synonymously with mobile devices. It should be noted that embedded devices are meant to represent all those devices with computational power less than 64MHz that are connectible to the outside world through Internet, either directly through GPRS or indirectly through Bluetooth, IEEE 802.11b or other technologies.

[2] The statement is supported by the fact that most embedded devices which allow third-party application programs provide a Java Virtual Machine – this is considered the norm here and exceptions are ignored.

made use of primarily for routine tasks and background “batch” processing. In this mode, HCG defines the computer-mobile as a node with a single *External Request Manager*. The grid may consist only of the computer-mobile combination[1], or may be connected to further computers on the network or internet. In the latter case, the computer is known as a *gateway*.

2. When network bandwidth is high enough, with technologies such as Wi-Fi, WiMax [21] or WiBro[2] [22], the embedded device may connect directly to the internet. In this case, the embedded device becomes a class III (Refer Figure 2) node connected to the internet with specifications such as “t.#3.32&osl.symbian68-java.13&nw.gp-9” or “t.#3.56&osl.wince2x-vbceljava.12&nw.wf-64”. This model allows the embedded device to contribute to the grid as also derive resources from the grid. In this model, the embedded device becomes a node of the grid like any other system. This has vast advantages for the grid and the embedded device.

3. The third method involves a supporting computer, called a *client scheduler* and is conceptually similar to method 1. The grid does not know about the inclusion of the embedded device, and only the supporting computer has information about the capabilities of the embedded device. The client scheduler divides the work equally between the host machine and the array of embedded devices, each of which has a unique registered *Device Access Identifier*. The work done is collated and transmitted to the grid as the total of the work done by the client scheduler. This has its advantages in that work can be done even when the client scheduler is switched off and the embedded device is idle, recharging its batteries.

Communication Protocols to be used

The HCG specifications are elucidated in the ‘Grid Network Protocol Usage Document’, the salient features of which are listed in Appendix B. Some of the major communication protocols used are given below.

Table I. Communication Protocols used

Protocol	Network	Port	Data transfer
UDP	LAN	above 1024	As messages: with callback for important messages.
HTTP	Internet	80	Input file and program as files to be downloaded. Result uploaded to location (or) sent as POST data.
WAP	Internet	-	Input file and program as applets/files to be downloaded. Result sent to file as form POST data.

Additional Constraints

The HCG specification defined several additional features, the most significant of which are given as follows:

1. The database must be relational. Object oriented and object relational data models are not recommended. The hierarchical and network data models may also not be used.

[1] A computer (or a set of computers) and an embedded device (or a set of embedded devices) can form a grid, in themselves, in which the embedded device is the problem poser and the grid resource consists only of the computational resources of the computer (or computers) connected to it. This is primarily implemented through Bluetooth or serial data cable.

[2] WiBro stands for Wireless Broadband, expected to provide up to 18Mbps download rate are all set to hit the South Korean markets by mid-2006. WiMax is based on the IEEE 802.16 working group recommendations.

2. It is recommended that the database on the server be architecture independent. In the simplest case, the database may range from a comma-separated file to a Microsoft Access database, and the database may be as complex as an Oracle database. The database used should be standardized and can be from among the following databases[1] – Microsoft SQL Server, Microsoft Access (97 and above), Microsoft FoxPro, Microsoft Excel worksheet (3.0, 4.0, 5.0 and 7.0), dBase (III, IIIB, IV and 5.0), Paradox, Microsoft Oracle ODBC, Comma-separated values (CSV), Sybase® SQL Server and Oracle Server (7, 8, 8i, 9i, 10g). Only databases from the above list must be used.

3. It is recommended that all integrity constraints as mentioned in Appendix A be enforced at schema level for all tables in the database. In case the underlying database driver does not support integrity constraints (such as a CSV or Excel driver), the constraints must be enforced at the middle-tier or the front-end itself.

4. While it is recommended that the database be accessed only through ODBC or JDBC drivers, an exception may be made for monitoring authority and debugging, benchmarking and performance tools, which can be allowed to access the database natively.

5. Requests from remote programs to read/write to the database must be authenticated by an additional layer of user code, before being passed on to the database. The database must not be required to authenticate change request through predefined roles. All changes to the database must pass through (and be verified and validated by) the server machine only. The remote user should not be given direct access to the database.

IMPLEMENTATION

Prototype Model

During the later stages of design of the Heterogeneous Computation Grid (HCG) framework, work started on the proof-of-concept phase and implementation phase. During this period of continuous modifications to the design document of HCG for optimal performance, five major prototypes of the grid were built and tested, as described in further detail below.

Choice of features

- Prototype I was a model for a peer-to-peer four-PC Windows 2000/XP-based systems from configurations ranging from Pentium III 500MHz to Pentium IV 2.8GHz HT over a 100Mbps LAN connection, with projects implemented as a shared folder "hcgprojects" on every PC, over a range of projects. The first prototype enabled the authors to make a comprehensive study on the feasibility of the projects and turned up several interesting results. Primary among them was the need felt for a central benchmarking utility, and the necessity of assigning priorities.

- Prototype II was the first prototype in which PCs in different platforms reacted with each other, through Java. Unlike the previous stage, wherein each program was made in Visual Basic® 6.0, the new project switched over entirely to Java™. Several cross-platform issues were resolved in the process.

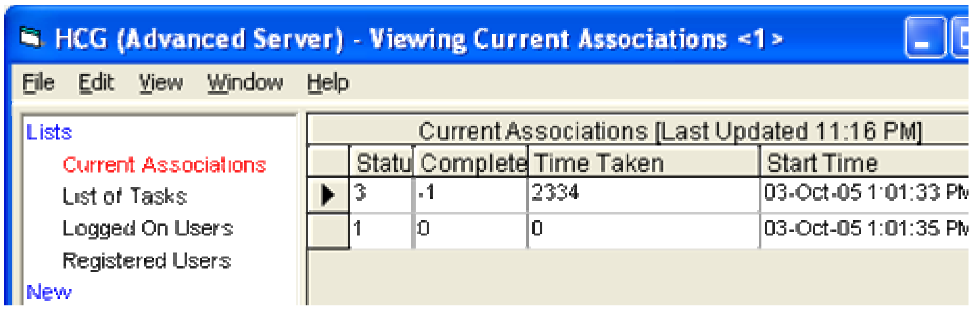
Use of Server

[1] The list of supported database may be increased at a later moment. All databases must be accessible via ODBC, JDBC or ISAM.

- Prototype III was an experiment to use several executables for the same task. Clients and Managers were separately built in Visual Basic 6 for Windows and in Java for Linux. Programs to be executed were offered in C++, Visual Basic and Java. The project on sandboxing was taking up during this stage when the potential damage that could be caused by a malfunctioning (malicious intent has been discounted here) program was realised. The prototype was rebuilt with a server (a Pentium IV 2.4GHz machine) and the number of PCs was increased to 6. The server managed the database and the load on the server database (initially MS Access 2003 and later changed to Oracle 10g [23]) was measured for varying loads on the grid. Benchmarking was also done in this stage.

Figure 3 shows a section of the server window with work in progress. The back-end database is Oracle 10g Release 1 and a ActiveX Data Object (ADO) has been used to connect using ODBC to the data source name (DSN) representing the Oracle database. The front-end is Microsoft Visual Basic 6.0 Enterprise Edition.

- Prototype IV came up with an attempt to extend the project to the internet. On the internet, the failure of direct UDP and TCP approaches led the authors to move to the application level of abstraction – HTTP. The server communicated a list of projects to the PCs by means of a XML file on the server. The XML file (which, incidentally, could also double up as an RSS feed) contained information about the various projects and their usage. 14-31 PCs connected to the internet participated in the 11 trials, with varied internet loads and speeds and several projects of varied scales were run on the grid. The results surpassed all expectations. For the first time, the tasks in each project were classified into three categories based on computational power (**C**), input problem and result size (**S**), and priority (P). Problems with large S and small C were given to LAN nodes, with large C and small S were distributed among internet users and problems with small S, small C and lower P were reserved for embedded devices and devices on the lower end of the performance spectrum.



Current Associations [Last Updated 11:16 PM]

	Statu	Complete	Time Taken	Start Time
▶	3	-1	2334	03-Oct-05 1:01:33 PM
	1	0	0	03-Oct-05 1:01:35 PM

Figure 3. Screenshot of HCG Advanced Server database viewer (Prototype III)

Client Executable

- Prototype V saw a major change in the structure of clients, with primary grids consisting of an array of grids, rather than an array of systems. Windows XP was chosen as the primary platform for the goal, being available on each of the 28 PCs used for the trial. Both LAN and Internet were used with the task categorization as described above. Java and Visual Basic code for the problems were given. Applets and ActiveX controls were used for the first time and were uploaded to a geocities server with large HTML files, primarily large novels. The results noted allowed several changes to the design specifications. Prototype V had completed 9 sets of trials, of maximum duration 26 hours, at the time of this paper.

Projects Run

The list of major projects solved or underway is given in Table II[1]. Some of these projects were used for testing purposes only and may not be completed. Some minor projects were used just for testing purposes. These included pattern searching, among other projects.

Table II. List of projects run

Prototype Number	Language	Project	Comp-leted?	Time on the project
I, II, III, V	VB, Java	Generating large prime number	N/A	Total 37 hours
II, III, IV, V	Java	Factorizing product of two large primes	Yes[2]	Total 22 hours
IV, V	VB	Mutiple dummy pre-generated floating-point operations.	N/A	Total 9 hours
IV, V	VB *etc.*[3]	Brute force for cracking 8-letter password.	Yes	Each 10-16 hours
V	C++	Solving dense system of linear equations by Gaussian elimination [24].	Yes	21 hours

Expected Additions to the Grid

More additions to prototype V may include a multi-platform trial and trial using embedded devices (a mobile phone was available for trials at the time of writing). Newer technologies such as sandboxing for non-java applications and code-signing are expected to be introduced by the time work for prototype VI is underway.

EVALUATION AND BENCHMARKING

The goal of this evaluation phase was to assess the performance of the grid under various scales of programs and conditions of network stress. The tests were done in local area network and internet environments. The results of the tests are summarized in Table III.

Table III. Test Results - Maximum Flops clocked

Project	Grid Characteristics	Max. Flops clocked	Remarks
Brute force for cracking password.	4 LAN PCs average 2.5 MHz	721M	2.4GHz – 2.8GHz range
	6 LAN PCs average 2.1 MHz	986M	500MHz – 2.8GHz range
	6 LAN PCs, 14 internet PCs	1014M	Internet speed maximum of 115Mbps over 6 PCs
	6 LAN PCs, 21 internet PCs	984M	
	4 LAN PCs, 28 internet PCs	833M	
	6 LAN PCs, 26-31 internet PCs[4]	991M	Small scale disconnections
	6 LAN PCs, 21-30 internet PCs	924M	Verifications compulsory
	4-6 LAN PCs, 8-22 internet PCs[1]	801M	Large scale disconnections

[1] It should be mentioned that the projects given here are not written with the most efficient algorithms as is generally the case in comprehensive testing situations. The goal of the testing was not to set benchmarks or solve critical problems but to root out potential flaws and areas for improvement in the design and implementation of the grid.
[2] The project was run 7 times, 2 of which gave the results and 5 of which were cancelled.
[3] Components used included Visual Basic 6 executables, Visual C++ 6 executables, Visual Basic 6 ActiveX controls, Java Applets and Swing applications, being the most comprehensive test case as of the time of writing of the paper.
[4] PCs on the internet are allowed to disconnect without informing the result or progress of the calculations.

Project	Grid Characteristics	Max. Flops clocked	Remarks
	6 LAN PCs, 31 internet PCs, 16 Applets	**1071M**	**Highest clocked**
	6 LAN PCs, 18-29 internet PCs, 4-9 Applets, 1-3 ActiveX controls[2]	1008M	Assumes LAN PCs are always available.
	6 LAN PCs, 30 internet PCs, 8 ActiveX controls	1044M	
	26 Internet PCs, 6 Applets, 11 Applets	621M	No PCs available on LAN
	6-28 Internet PCs[3]	540M	
	4 ActiveX controls, 3 Applets	320M	
Solving linear equations	6 LAN PCs, 31 internet PCs	1013M	

The test results in Table III have the following implications:

- For the password-cracking problem with small S and moderate C, the presence of large number of internet PCs does not significantly increase the computational power of the grid. The grid works best when there each task has small problem definition, an easily verifiable answer, and requires lots of computations for solving the problem.

- Test results show that a single Pentium IV 2.4GHz PC gives a computational power of over 200MFlops for native code on Windows. Allowing for losses, 6 PCs should easily gross 1100MFlops. The balance represents the loss due to network transmission, which can be minimised by reducing S and increasing C.

- A Windows GUI executable compiled to native code, optimized for speed, in Visual Basic 6.0 running on Windows XP, with SP2 installed grossed 228MFlops on a lightly-loaded Pentium IV 2.4GHz machine. Under the same conditions, a Java CLI application clocked 146MFlops maximum when run with Sun Microsystems' JRE 1.5. This is a speed loss amounting to nearly 35 percent. Therefore, as far as possible, it is best to avoid interpreted languages such as Java, when native executables for other platforms are available. The programmer workload has be balanced with faster execution speeds and this decision can be taken best only by the distributed problem developer.

CONCLUSIONS

The paper has described a simple, economical and heterogeneous grid system that has been developed, tried and tested for large-scale simulation studies with computing resources distributed over a large set of networks – the LAN an the Internet. Heterogeneous Computation Grid makes use of existing hardware to connect all possible devices and create a shared pool of enormous computing power. The grid is self-administering and works on an equal-priority basis. The grid does not make any restrictions, on basis of architecture, operating system, network connectivity and so on. Unlike major grid solution companies, the independence between the grid and the user program running on it has been maintained. The system has displayed excellent performance in terms of robustness, and throughput and new features are to be added, further improving the grid and enhancing its security. Further, the simplicity

[1] Both local networked PCs and internet PCs are allowed to disconnect, requiring work to be re-assigned and resolved. The limits given are the minimum and maximum number of systems on at any moment, and not the total count of systems.
[2] The most realistic close-to-practical expectations test case.
[3] Only internet PCs were used and problems with large C and small S were given.

of recommendations made in the specification make the grid highly scalable and extendible, which will be an contributing factor in its acceptance by the distributed computing community.

ACKNOWLEDGEMENTS

The authors thank all those students of their institute, who donated idle CPU time of their machines and network bandwidth for the development and testing of a prototype for this paper.

APPENDICES

APPENDIX A
Database Schema for Server – Summary

The compulsory fields required by HCG are given in Tables IV and V, while describe the tables defined by HCG, and may be used if required.

Table IV. Schema of Table "users"

Field Name	Data type	Constraints	More Information
userid	`VARCHAR2`	`PRIMARY KEY`	User / Machine Identification
priority	`NUMBER(1,2)`	`NOT NULL`	Range 1 (lowest) to 9.99 (highest); default is 3
loggedon (optional)	`DATE`		Last log on date/time. User logged on if date/time within session expiry time, say, 20min.
pctype	-	`NOT NULL`	

Table V. Schema of Table "projects"

Field Name	Data type	Constraints	More Information
projectid	`LONG`	`PRIMARY KEY`	Project ID Number (auto-generated)
pname	`VARCHAR2`	`UNIQUE`	Project descriptive name
dispatchers	`VARCHAR2`	`NOT NULL`	List of dispatchers addresses
acceptors	`VARCHAR2`	`NOT NULL`	List of acceptor addresses
author `(REF)`	`VARCHAR2`	`NOT NULL`	Must exist in the users table in "userid"
priority	`NUMBER(1,2)`	`NOT NULL`	Always less than user's priority, the default.

APPENDIX B
Grid Network Protocol Usage Document – Summary

The salient features of the *Heterogeneous Computation Grid Network Protocol Usage Document* are listed below:

1. Communications between PCs on a LAN takes place through User Datagram Protocol on any pre-registered port greater than 1024. A system can be a part of two independent HCG and use different ports for both of them. The port cannot be negotiated, but must be specified at the time of setting up the grid. Port numbers have to be specified manually

2. Since UDP communication carries little overhead, and is fast but unreliable, all messages to be passed with a time indication of seconds since previous hour and a hash of the current day/hour pair. All UDP message must contain a two-byte repeated message type identification (MTID) which is 1 for "Call-back Required", 4 for "Encrypted Message", 8 for "Critical Message" and

so on; and a 4-byte hash value. Messages with odd MTID must be replied back with acknowledgement. Additionally, for encrypted messages, the reply must be encrypted as well.

3. On larger networks such as a WAN or the internet, UDP is so unreliable that it cannot be used. Here, application layer TCP/IP protocol Hyper-Text Transfer Protocol (HTTP) is the protocol of choice for transferring data. Program and task-data locations are dispatched by eXtended Markup Language (XML) files. Programs that must be transferred to the client PC are downloaded over port 80. Results are sent to the acceptor as form POST submissions. The acceptor, as a simple server script, accepts/results the results and returns an XML file, format of which is predefined.

4. Files transferred over the internet may be digitally signed. Client policies can dictate the download and execution of unsigned code. Execution may also be limited to a sandbox environment, if available for the target executable format. The client is free to enforce such policies for code over the internet.

5. The client when sending back results must encrypt it with the public key of the author, if available.

6. It is recommended that all communications for the LAN and the internet use these protocols only. This will enable grid developers to use servers and clients made by the authors, and enable the use of parallelizing grid-access code used to program the grid, also provided by the author. However, if specific working conditions of the grid mandate it to deviate from the above specification for optimum performance, the grid developers can still do so while remaining within the Heterogeneous Computation Grid framework. However, client and server design may need to be changed to work with the new grid.

7. Embedded devices may communicate with the grid in more than one ways. A clear definition of or limitation on the protocols in use by embedded devices is not available. Since Heterogeneous Computation Grid uses primarily existing standards and frameworks for establishment of a low-cost high-performance grid and no current single standard exists for embedded device communication with a computer (or a computer network), the embedded device communication protocols have not been defined in the document.

8. The above rules are to be interpreted as advisory guidelines for the design and development of any peer-to-peer heterogeneous grid.

REFERENCES

[1] B. Hayes, "Collective Wisdom", American Scientist Online – COMPUTING SCIENCE, March-April 1998.

[2] D. P. Myers and M. P. Cummings, "Necessity is the mother of invention: a simple grid computing system using commodity tools", Journal of Parallel and Distributed Computing Vol. 63, pp. 578-589, 2003.

[3] The distributed.net homepage, http://www.distributed.net/

[4] The SETI@home project homepage at the website of the University of California, Berkeley, California, http://setiathome.ssl.berkeley.edu/

[5] The Berkeley Open Infrastructure for Network Computation (BOINC) homepage, University of California, Berkeley, California, http://boinc.berkeley.edu/

[6] D. J. Becker, T. Sterling, D. Savarese, J. E. Dorband, U. A. Ranawak, C. V. Packer, "Beowulf: a parallel workstation for scientific computation", Proceedings of the 24th International

Conference on Parallel Processing, Oconomowoc, Wisconsin, pp. 11-14, 1995.

[7] I. Foster and C. Kesselman, “The Grid: Blueprint for a Future Computing Infrastructure”, Morgan Kaufmann Publishers, Los Atlos, California, 1998.

[8] I. Foster and C. Kesselman, “Globus: a toolkit-based Grid architecture”, in I. Foster and C. Kesselman, “The Grid: Blueprint for a New Computing Infrastructure”, Morgan Kaufmann Publishers, Los Atlos, California, pp. 259–278, 1999.

[9] The Globus project homepage; http://www.globus.org/

[10] The Java™ home page on the Sun Microsystems website, http://java.sun.com/

[11] D. Abramson, R. Sosic, J. Giddy, and B. Hall, “Nimrod: A tool for performing parameterised simulations using distributed workstations”, Proc. 4th IEEE Symposium on High Performance Distributed Computing, IEEE Computer Society Press, 1995.

[12] J. Baldeschwieler, R. Blumofe, and E. Brewer, “ATLAS: An infrastructure for global computing”, Proc. 7th ACM SIGOPS European Workshop on System Support for Worldwide Applications, 1996.

[13] H. Casanova and J. Dongarra, “Netsolve: A network server for solving computational science problems”, Technical Report CS-95-313, University of Tennessee, November 1995.

[14] G. McGraw and E. Felten, “Java Security: Hostile Applets, Holes and Antidotes”, John Wiley and Sons, Inc., New York, 1996.

[15] Verisign Code Signing and Digital Identification Homepage, http://digitalid.verisign.com/

[16] I. Goldberg, D. Wagner, R. Thomas, and E. Brewer, “A secure environment for untrusted helper applications”, Proc. 6th Usenix Security Symposium, July 1996.

[17] D. Wallach, D. Balfanz, D. Dean, and E. Felten, “Extensible security in Java”, Technical Report 546-97, Dept. of Computer Science, Princeton University, 1997.

[18] A. Silberschatz, J. Peterson, and P. B. Galvin, “Operating Systems Concepts”, Addison-Wesley, 1991.

[19] Homepage of Java 2 Micro Edition for mobile platforms, http://java.sun.com/j2me/

[20] The official Bluetooth homepage, http://www.bluetooth.com/

[21] News item on “WiMax and Wi-Fi: Separate and Unequal”, IEEE Spectrum INT, March 2004

[22] News item on “South Korea Pushes Mobile Broadband”, IEEE Spectrum INT, September 2005

[23] The Oracle 10g homepage http://www.oracle.com/database/

[24] The Linpack reference page at the NetLib website, http://www.netlib.org/linpack/