

# Supporting Peer-2-Peer Interactions in the Consumer Grid

Ian J. Taylor, Omer F. Rana, Roger Philp<sup>1</sup>, Ian Wang<sup>1</sup>, Matthew Shields<sup>1</sup>,  
Department of Computer Science,

<sup>1</sup>Department of Physics and Astronomy,  
Cardiff University, UK

## Abstract

*A “Consumer Grid” provides the individual-based counterpart to the organisation-based computational Grid. We describe a Peer-to-Peer system for utilising computational resources on the Grid – extending existing work undertaken in systems such as Entropia and SETI@home. The potential of such a distributed computing resource has been in some ways demonstrated recently by the SETI@home project, having used over 650,000 years of CPU time at the time of writing. A user develops applications within such an environment using a visual workflow system called “Triana” – which automatically generates suitable code for distribution, and can support the user in making placement decisions for their modules. Triana will also be deployed as the workflow enactment engine along with the Grid Application Toolkit (GAT) within the European GridLab project.*

## 1 Introduction

The Internet provides a fast growing resource capable of a vast amount of CPU power if connected in an organised way. Recent results from demographic studies by the Computer Industry Almanac [1] reports an expected 490 million Internet users by year-end 2002 and over 765 million by year-end 2005. Current reference solutions for Grid computing [2] have addressed this problem partially by allowing organisations in widespread locations to be connected in a secure way to create computational Grids. However, this does not address what we consider the “Consumer Grid” [13], that is, a Grid environment based on users that are potentially permanently connected to a network via cable, DSL or similar but which do not belong to an organisation. Such users are likely to provide a majority of the usable Grid resources, and CPU power that they can offer has yet to be adequately utilised in both the scientific and business computing arenas. We present a construction tool for developing Peer-to-Peer (P2P) systems and applications, called Triana, that can graphically build distributed programs that can be easily deployed onto the Consumer Grid. It is important to emphasise at this point, however,

that the Consumer Grid is intended to target resources such as DSL/Cable, and the variety of devices that can be connected together using these technologies – comprising of, primarily, privately connected individuals. Such resources do not belong to an organisation as such and therefore are currently not targeted by Grid research. The Consumer Grid therefore is a complimenting paradigm to existing Grid technologies and not a competitor.

Our vision of The Consumer Grid is aimed at providing the capability to bring Grid technology to the masses (the variety of individual users), and enable truly distributed computing over public networks. We assume that the user has access to the executable code (in the form of Java classes), which they can execute on their own resources and can be transferred to the node where the execution is to be performed. P2P computing and Web services ideas have been explored by other authors, such as [3], where ideas related to computing portals, and a common software component architecture (CCA) have been outlined. End users who program applications using pre-packaged software are likely to be the most prolific users of Grid infrastructure. The development of interfaces to enable such users to utilise existing software libraries therefore becomes crucial, especially if such interfaces can be accessed across a network. Such users would need to know very little about Grid infrastructure, protocols or services directly. The Web infrastructure (Web servers, client browsers) has provided a useful first step to realising such interfaces, although the complexity of services that can be supported is limited. This is shown by the lack of semantics that can be captured in an HTML document, necessitating the development of specialised meta data standards based on XML. Examples of Grid portals currently in use include XCAT [4], Gateway [5], and GridPort [6].

A useful abstraction to adopt in the context of a Consumer Grid is the notion of a “service”. We can consider the availability of all computational resources on the Grid as a type of service. Furthermore, within P2P systems, every entity on the network can be both a service user and a service provider. In this context, a Consumer Grid is composed of a number of peers. Each peer provides a service that is

analogous to a Web server, in that it can receive and process requests and returns results. For example, a Web browser (client) will make an HTTP request to a Web server, and the Web server utilising a number of server-side programs, will return a result back to the client, which will be rendered according to the Web browsers capabilities. There are also similarities between this model and the Open Grid Services Architecture (OGSA) [12], in that both utilise the service abstraction. However, the Consumer Grid makes use of interacting peers, rather than the client-server model adopted in OGSA.

In our reference implementation, the Consumer Grid is composed of a number of Triana peers. Triana is a service, which utilises services from other peers to schedule applications across a network. A Triana peer receives requests in the form of Triana scripts and Triana data, processes these on the local computational resources, and returns results back to the Triana client or onto another peer. These requests are encoded as XML scripts, and outline various attributes of the host on which the service is to be executed, and the execution interface for the service. We also hope to provide a Web Services Description Language (WSDL) interface to these at a later time, through the Java2WSDL interface from IBM [14]. In the same way that an Applet has security on the client side, we provide a similar level of security on the Triana server through the Java Sandbox. The Java Sandbox comprises a number of cooperating system components, ranging from security managers that execute as part of the application, to security measures designed into the Java Virtual Machine (JVM) and the language itself. The sandbox ensures that an untrusted and possibly malicious-application cannot gain access to system resources [8]. Furthermore, a Triana server could be implemented as a Servlet and run as a Web service.

## 2 Interactions with Globus

A current system for deploying Grid based applications is Globus [2] [11], which provides a means to securely connect organisations together by offering a single sign-on method for accessing all resources within an organisation's network. Globus uses public and private keys obtained from a, hopefully trusted, third party certificate agency (CA) that allows for secure sign-on and data transfer methods to resources. Globus at present provides access to CPU resources primarily on UNIX platforms although some of the client-side tools do exist for PCs running a non-UNIX based operating system such as Windows: an operating system adopted by the majority of computer users.

Although Globus has a sophisticated level of authentication, it is in many ways similar to standard telnet, ftp and other job submission methods currently in use. After suitable certificates and keys have been obtained, and an ac-

count requested from an administrator of a resource, one has to sign on to the Globus environment and explicitly target a resource for job execution. This normally occurs through the batch job scheduler, although access to an interactive session can be made through the use of shell scripts and the fork process.

Some of the difficulties with Globus and perhaps some of the biggest drawbacks associated with it, is the administration cost and difficulty of use. Administrators with resources that they are willing to make available have to create accounts explicitly for Globus users. If thousands of users wanted access to a resource it would be a daunting task indeed for any administrator. The use of an "anonymous" Grid account is supported on other Grid systems, such as Legion, but not on Globus. It is also likely that users may utilise a machine infrequently, making the administrative overheads of creating and registering accounts a significant overhead. However, the problem of job interference (and security issues) with limited number of Grid accounts is another issue that we are currently investigating in the context of the Consumer Grid. This functionality would perhaps be best served by the creation of a single Globus account, with an associated Globus shell (similar to csh or tcsh) for the resource and a daemon informing the CA of the resources available. The shell would also maintain billing information for resources used.

Triana, however, does not implement this level of authentication and is platform independent – as it is implemented in Java. Triana Services can be run on various hardware and operating system platforms that support a Java Virtual Machine. This implicitly means, like the SETI project, that anybody can make their spare CPU cycles available. It installs easily with a "point-and-click" method to instantiate a service daemon. Triana does not rely on Certification Agencies: once a simple service daemon has been installed on a platform it will allow access to that resource. Program modules are automatically transported and executed on resources enrolled in the Triana environment effectively using a virtual account. As Triana is Java based, it makes use of the Java Sandbox, and resource file systems are also automatically protected. Triana can seamlessly distribute modules and entire jobs across a network of compute resources allowing a Triana network to perform High Performance Throughput (HPT) calculations, parameter space searches, Parallel computation, and even behave as a macroscopic pipeline processor where one machine performs one specific task and then pipes data onto another machine.

At present, the Grid assumes that participating users are trusted (or that they have obtained a certificate from a trusted authority – as discussed above). This is generally a valid assumption, as existing sites offering Grid enabled software are large research centres or institutions and can be held accountable if they refuse access to their re-

sources once they have committed. To make Grid technologies more widely usable, however, such an assumption may not hold. Furthermore, where service providers do not belong to one of these large sites, managing and running services over large and complex servers may not be an option. Hence, if individual scientists were not allowed to offer software libraries that they had implemented, the Grid would be restricted to specialised service providers, and is unlikely to have a large impact on the scientific and business community. We therefore see the need for finding a synergy between the Grid infrastructure as it is being implemented at present, and a more generic infrastructure that can be more easily shared and deployed.

The P2P approach provides a useful way to overcome this limitation – a P2P network is a type of network in which each connected device is able to communicate and collaborate as a peer. Such infrastructure has already demonstrated scalability to over 3 million users world-wide (in the case of Napster), although at present such tools target only a particular kind of service (e.g. the availability of MP3 files).

### 3 Triana

#### 3.1 The Triana Software Environment

Triana [17] is a Java-based problem-solving, data-analysis and programming environment, which allows the composition of various network Peers to collaborate to solve a single problem. It provides a visual programming environment allowing a user to construct an application by connecting different Peer services. It comes with many built-in functions that can be used to manipulate numeric, signal, image and textual data – and also provides a set of built-in data types that can be used to connect different Peer services – and undertake type checking on their connectivity. There are several hundred units (i.e. programs) and networks of units can be created by graphical connections to construct new and more complex programs.

Triana is a two-layered application consisting of a Triana engine and a graphical user interface (GUI). The Triana GUI uses the Triana engine to connect objects together to construct data-flow networks. A Triana network can be constructed using the GUI or directly by writing an XML task-graph (in Web Services Flow Language (WSFL), Petri net or Business Process Enactment Language for Web Services (BPEL4WS) formats). An application example is illustrated in Figure 1. The figure illustrates a simple network that creates a sine wave, contaminates it with Gaussian noise, takes its power spectrum and then uses a unit called `AccumStat` to average the spectra over successive iterations to remove the noise from the original signal. In figure 2 we show two outputs, one taken after the first iteration (notice that the signal is buried in the noise) and the other after 20 iterations of

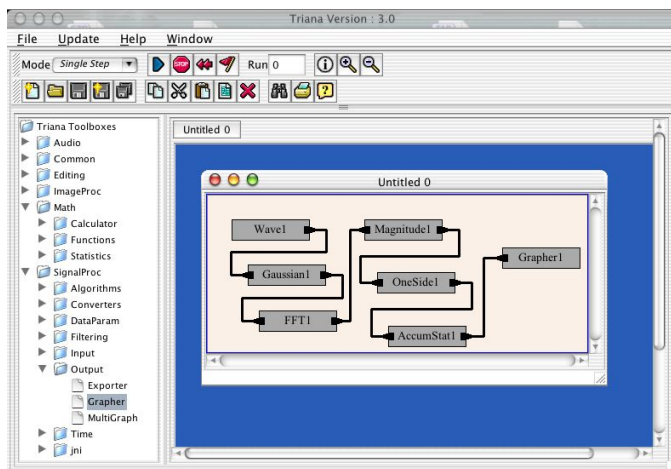


Figure 1. Triana user interface

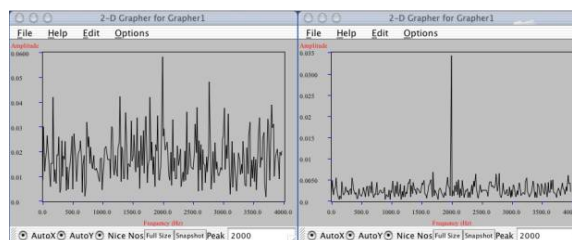


Figure 2. Triana graph output component

the algorithm.

Each Triana node requires the execution of both a client component and a server component – hence the notion of a Triana Peer. A user may connect to a Triana Peer service using a command line or GUI interface. The client part of a Triana service can distribute code to many servers, depending on where execution is required. A Triana network is composed of a collection of Peers, each capable of interpreting Triana XML task-graphs. The server component within each peer can interact with Globus GRAM to launch jobs locally on the node. This is useful to support nodes which host parallel machines or workstations clusters. A Triana network therefore can be composed of a number of different kinds of resource management systems – supported via a gateway between a Triana Peer and the particular system used to launch and manage jobs. The complexity of such local systems can vary, depending on the capability of the device hosting the Triana Peer. In the case where no local resource manager is available, the Triana server component can itself be used to launch the application.

The distribution mechanism used in Triana utilises an on-demand download of code – if this feature is supported within the local resource manager on the device. Hence,

when distributing an application, a Triana peer can send a connectivity graph to another peer node within the network. This peer must now decide whether the connectivity graph it has received should be executed locally, or whether all or parts of it should be submitted to other peer(s). If the graph is to be locally executed, the peer can request executable code for modules that are present within the connectivity graph. This dynamic download of code, depending on what is to be executed by a peer, allows the peer to only host code that is necessary – and overcomes the problem of having inconsistent versions of executables (as the executable must be requested from the owner whenever an execution is to be undertaken). This model is also useful when a particular device has limited capability to host code locally – due to memory constraints for instance. A resource-constrained device may also decide to selectively download and release executable modules based on dependencies inherent within the connectivity graph. This dynamic model is therefore particularly useful for handheld and mobile devices – and may be used to support the development of Triana Peer networks where devices download and release code modules on-demand. Transmitting the connectivity graph to nodes has a limited overhead – as the graph itself is a text file that does not consume many resources.

### 3.2 Triana implementation of a Consumer Grid

To support the consumer Grid, the Triana implementation disconnects the user interface from the Triana engine. Communication from the user interface is via a defined API to the Triana engine that can be accessed by other views of the Triana network. For example, the reference graphical user interface provided in the Triana distribution may be the one of choice by software developers working on a desktop or laptop computer, but users may want a different view when utilising a WAP enabled mobile phones or PDA device. Furthermore, users should be able to obtain progress of their running network via the internet using a standard Web browser. The description of a Triana unit is also encoded in XML, and based on the the Common Component Architecture (CCA) [20].

As seen in figure 3 there are two distinct components in the Triana implementation: the Triana Service (TS) and the Triana Controller (TC). The Triana controller is a user interface to Triana service daemons. The Triana controller can be based either on a command line or a GUI user interface, and the Triana service daemon may be either local or remote. The Triana controller provides access to a network of computers running Triana service daemons via a gateway daemon and allows the user to describe the kinds of functionality required of Triana: the module deployment model and data stream pathways. A single Triana controller can control multiple Triana networks deployed over multiple

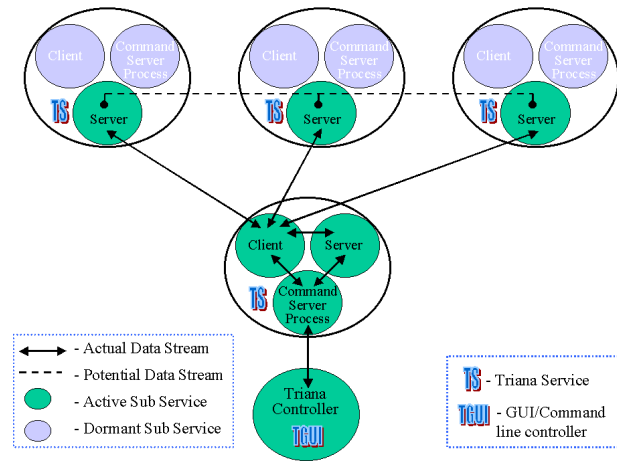


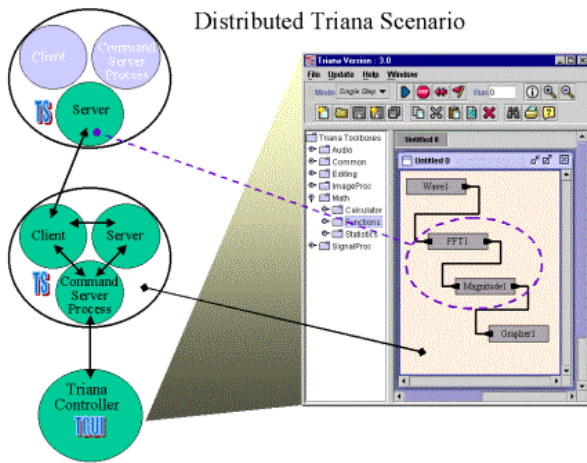
Figure 3. Shows a schematic representation of the Triana implementation.

CPU resources. The Triana Service daemons have multiple functionality and may be downloaded from a pre-defined portal. Since Triana is implemented in Java, the Triana Controller and Triana Services can be installed on any platform that supports a Java Virtual Machine. The Triana Service is comprised of three components: a client, a server and a command process server. Typically in a Triana network only one of the Triana services command process server will be active and in communication with the Triana controller – which acts as a scheduling manager for the complete application being run over a Triana network. The client of the Triana service in contact with the Triana controller then pipes modules, programs and data to the other required Triana service daemons. These Triana services will simply act in server mode to execute the byte code and pipe data to others via the prescription given by the Triana controller.

Figure 4 shows the implementation of the GUI version of the Triana Controller. It also shows the pallet upon which modules may be placed and how data flow is achieved by linking modules together: in this particular case a simple distributed pipelined linear network has been created. The figure also shows how a Triana Service provides communication to the Triana Controller and also how it also passes module information onto the server side of other Triana Services running on remote machines.

### 3.3 Distribution Mechanisms Within Triana

We imposed several design constraints for the distribution mechanism within Triana. Firstly, the visual representation within the TGUI should hide the complexity from the user. Secondly, we wanted the implementation to be mid-



**Figure 4. Illustrates the actual implementation of the Triana model.**

Middleware independent i.e. mapping to alternative middleware architectures should be trivial. Thirdly, we wanted a flexible enough approach to allow easy extensibility of new user distribution policies.

The current implementation meets these three constraints. It is based around the concept of Triana Group units. Group units are aggregate tools which can contain many interconnected units. They have the same properties as normal tools e.g. they have input/output nodes, properties etc, and therefore, they can be connected to other Triana units using the standard mechanism. Tools have to be grouped in order to be distributed allowing single or groups of units to be distributed onto the Grid i.e. the user has the complete control of choosing the desired level of granularity. However, the way in which groups can be distributed is extremely flexible.

Each group has a distribution policy which is, in fact, implemented as a Triana unit. Such units are called control units. A distribution policy is the mechanism for distribution within a group and therefore there is one control unit per group. This flexible approach means that it is easy for new users to create their own distribution policies without needing to know about the underlying middleware or specifics about individual Triana units. There are two distribution policies currently implemented in Triana, parallel and peer to peer. Parallel is a farming out mechanism and generally involves no communication between hosts. Peer to Peer means distributing the group vertically i.e. each unit in the group is distributed onto a separate resource and data is passed between them. Control units reroute input data and dynamically re-wire the task graph to create a distributed version that is annotated with the particular re-

sources the particular groups will run on and the specific data channels that are used for the communication.

### 3.4 Triana P2P Consumer Grid Implementation using JXTA

Triana currently has the ability to distribute a single unit, or any collections of its units amongst a set of distributed computers. This is currently based on the JXTA architectures for distributing and locating available peers through resource discovery. The full implementation details of this is available from [17] but briefly described below.

The JXTA mapping uses the so-called *JXTAServe* API to implement the mapping between Triana groups and Triana resources. *JXTAServe* is an API developed as part of the Triana project to enable users to build applications using the JXTA protocol. It implements the basic functionality that an application needs and hides the complexity of the details of JXTA from developers. Furthermore, since JXTA is an evolving system *JXTAServe* provides the stability for our implementation in Triana i.e. even though JXTA may change, the interface to *JXTAServe* will not. This protects the Triana developers from needing to rewrite the core code for each new JXTA release. Eventually, *JXTAServe* will be merged into the GAT (Grid Application Toolkit). The GAT is beyond the scope of this paper, for more details see <http://www.gridlab.org/>.

*JXTAServe* therefore implements a service-oriented architecture based on JXTA. It conceptually looks very similar to a Triana group unit. A *JXTAServe* service can have one or more input nodes (one is needed for control at least) and can have zero, one or more output nodes. It advertises its input and output nodes as JXTA pipes and connects between pipes using the virtual communication paradigm in JXTA networks. This adapts to the particular communication protocol which lies beneath the application depending on the situation.

Triana services are run as *JXTAServe* services and their input and output nodes are advertised as *JXTAServe* input and output pipes. There is almost a one to one correlation with the Triana implementation and the functionality of *JXTAServe*. This is one use of *JXTAServe*, however, the norm is very different.

```
<tool>
  <name>GroupTest</name>
  <tasks>

    <task>
      <taskname>Wave</taskname>
      <output> triana.types.SampleSet </output>
      <parameters>
        <param name="intensity"
          type="userAccessible" value="1"> ...
      </parameters>
    </task>
```

```

<task> <taskname>Grapher</taskname> ... </task>

<task>
  <taskname>GroupTask</taskname>
  <tasks>
    <task> <taskname>FFT</taskname> ... </task>
    <task> <taskname>Gaussian</taskname> ... </task>
  </tasks>
  <connection>
    <source taskname="Gaussian" node="0" />
    <target taskname="FFT" node="0" />
  </connection>
  <groupnodemapping>
    <input> <node externalnode="0"
      taskname="Gaussian" node="0" /> </input>
    <output> <node externalnode="0"
      taskname="FFT" node="0" /> </output>
  </groupnodemapping>
</tasks>
</task>

<connection>
  <source taskname="Wave" node="0" />
  <target taskname="GroupTask" node="0" />
</connection>
<connection>
  <source taskname="GroupTask" node="0" />
  <target taskname="Grapher" node="0" />
</connection>
</tasks>
</tool>

```

#### Code Segment 1

Code Segment 1 provides a simplified example of a Triana workflow definition. This example workflow defines four tasks (a Wave generator task, Gaussian noise task, FFT task and a Grapher task) and the data flow connections between them (Wave to Gaussian to FFT to Grapher), and based on the connectivity in figure 1. Within the workflow the Gaussian noise task and the FFT task are grouped into a group unit (called GroupTask). Hence, rather than the Wave task being connected directly to the Gaussian task, it is connected to node0 of the GroupTask, and it is within the group that the mapping between node0 of the GroupTask and node0 of the Gaussian is made (in <groupnodemapping>).

As mentioned previously, in Triana the unit of distribution is a group. In terms of our workflow example we could execute the GroupTask on a remote Triana service, with the data being automatically sent from the Wave to the Gaussian and returned from the FFT to the Grapher. To initialise this distribution a workflow is annotated in two ways: firstly, each group input and output connection is uniquely labelled by the local service; and, secondly, the group being distributed is extracted from the workflow and sent to the remote Triana service. The initial unique labelling of the group's connection enables the local and remote services to map input/output pipes to each of these connections. In JXTA this is simply done using the standard JXTA pipe dis-

covery mechanism. Briefly, for each input connection, the remote service advertises an input pipe with that connection's unique name. Since the local service knows the connection's unique name it locates the pipe with that name and binds to it; this process works vice

### 3.5 Triana Availability and Deployment

Triana is being extended to implement its distribution of units amongst a set of virtual organisations by using the GridLab GAT (Grid Application Toolkit) API [22]. The GAT API defines a high level API to existing Grid systems, such as Globus, to enable services on these systems to be transparently invoked.

To deploy the Consumer Grid, a user would need to have the Triana peer installed locally. The peer acts as a hosting engine on a given resource which is to be used to execute an application – and it is assumed that this has been achieved prior to initiating the execution of the application. Additional user interface capabilities are provided through the Triana Controller GUI – which only needs to have a single instantiation for a particular application. A device user must therefore agree to participate in a Consumer Grid by allowing the Triana peer to exist on their computation resource. The user is therefore required to trust the Triana peer – although as indicated above, the sandbox provides some protection to the resource owner. However, this is the only security policy currently supported, although we hope to investigate the development of more complex trust models (and security policies) in the future. Another security aspect not addressed yet is the actual application that will utilise the Consumer Grid – for instance, although a user may agree to contribute their resources (by hosting a Triana peer), they would not have direct control of what application actually utilises their resource (this is the classic problem of being unable to prevent a collection of computational peer resource being used to crack passwords). This is a difficult problem to overcome – as it is possible for a user to disguise the computational tasks they distribute to peers – and therefore difficult to detect. An alternative approach would be to allow users to only download executables that are selected from a pre-agreed, certified, software library. We believe that the Triana peer environment will help us investigate some of these issues – by providing a test bed for experimentation.

### 3.6 Triana Application Scenarios

We assume that in the context of a Consumer Grid some service providers will continue to offer the same service during their lifetime, whilst others may change over time. We provide usage scenarios to demonstrate how services may be utilised:

### 3.6.1 Case 1: Galaxy Formation Example

As a test case for the Triana distribution policies described in the previous section, we integrated a Java galaxy formation code developed by the *Galaxy Formation Group* at Cardiff into Triana. The implementation used the parallel distribution policy for groups for farming out the individual sections of the animation.

Galaxy and star formation simulation codes generate binary data files that represent a series of particles in three dimensions, along with their associated properties as a snap shot in time. The user of such codes would like to visualise this data as an animation in two dimensions, with the ability to vary the perspective of view, and project that particular two dimensional slice and re-run the animation. Due to the nature of the data, each frame or snap shot is a representation at a particular point in time of the total data set. It is possible to distribute each time slice or frame over a number of processes and calculate the different views based on the point of view in parallel.

In the implementation the data file is loaded by a single Data Reader Unit within Triana, and passed to all the Triana nodes. Nodes then buffer the data for future calculations. Note that the data file could be copied beforehand and distributed in a parallel way also. The loaded data is then separated into frames, distributed amongst the various Triana servers on the available network and processed to calculate the column density using smooth particle hydrodynamics.

A Triana visualisation unit on the users local machine is used to control the entire process and run the animation once the processing is complete. Each distributed Triana service returns it's processed data in order, allowing the frames to be animated. If the user wants a different view of the data the visualisation unit has controls that allow the manipulation of the view, messages are then sent to all the distributed servers so that the new data slice through each time frame can be calculated and returned. The result is that the user can visualise the galaxy formation in a fraction of the time than it would if the simulation was performed on a single machine. This implementation was demonstrated successfully at the All Hands Meeting in UK in September 2002 using machines on a local network.

### 3.6.2 Case 2: Inspiral Search for Coalescing Binaries

Compact binary stars orbiting each other in a close orbit are among the most powerful sources of gravitational waves. The gravitational waves emitted in the process have twice the frequency of the binary and carry away its energy. This results in the gradual shrinking of the orbit. As the orbital radius decreases, a characteristic chirp waveform is produced whose amplitude and frequency increase with time until eventually the two bodies merge together. Laser in-

terferometric detectors, such as GEO600 should be able to detect the waves from the last few minutes before the collision [23].

For this search, we need to have a computing resource capable of speeds in the range of 5 and 10 Gigaflops to keep up in real time with an on-line search. For example, within GEO 600, the gravitational wave signal is sampled at 8kHz in 24-bit resolution (stored in 4 bytes). However, the searchable frequency range is below 1 KHz and therefore a realistic sampled representation of the signal contains 2,000 samples per second. The real-time data set is divided into chunks of 15 minutes in duration (i.e. 900 seconds), which results in a 7.2MB of data (4 x 900 x 2000) being processed at a time. This data is transmitted to a Triana node and processed locally. The node initialises i.e. generates its templates (a trivial computational step) and then it performs fast correlation on the data set with each template in a library of between 5,000 and 10,000 templates. This process takes about 5 hours on a 2 GHz PC running a C program. Therefore, 20 PC's would need to be employed full-time to keep up with the data.

Within a Consumer Grid scenario the number of PCs would need to be increased due to various types of downtime e.g. connection lost, user intervenes, computational bandwidth not reached etc. However, since it is a massively parallel problem we believe it can be solved within such an environment by simply distributing the code to as many computers that are available until the results are being returned with the specified time interval. The latency of such a system is not important and it can lag behind by several hours if necessary. A check-pointing mechanism may also be employed to migrate computation if necessary.

### 3.6.3 Case 3: Database access

A Triana user can also undertake database access by connecting components together – especially to enable multiple users to manipulate a database. In order to do so, the user establishes a pipeline in Triana consisting of: (1) a data access service, (2) a data manipulation service, (3) a data visualisation service, and (4) a data verification service. The data access service can either read from flat files, or read from a structured database. After the pipeline has been composed, the user provides preferences for the kinds of services that should be utilised in each of these cases. Each of these services may now be provided by different Triana Peers – which may be located at different geographic sites. A Peer providing a data access service, for instance, can communicate with a database using Java DataBase Connectivity (JDBC) bridge.

The Triana system looks on the network to discover peers which offer each of these services in turn. The pipeline is instantiated with peer references as new services

become available. The Triana controller now initiates execution of the entire network. If Triana finds multiple components to manipulate the data, the user may be asked to select a service based on other options that a given service provides (such as accuracy, numerical and textual data manipulation capability etc). Once a service has been selected, and the Triana system has undertaken a service-bind to each of the stages in the pipeline, Triana now initiates the execution procedure.

### 3.7 Availability of Peers?

An obvious question at this point is why Grid users would make their CPU available to others? Our belief is that users would altruistically make their computers CPU and RAM available if they trusted the software, whether they thought its use was a worthy cause and if they could have control about when their resource could be made available. An obvious solution to this is to take the approach that Condor [24] and SETI [25] take and make user's CPU available when their workstation is idle i.e. when the screen saver turns on, for example. Users also would have the option to specify how much RAM the applications could use and publish this on the network. Users would then run the software in the same way in which Napster or Gnutella users run their peers, but instead of sharing mp3 files they would be sharing their computational power with others who could make use of it. Current examples of sharing resources are:

- **SETI@home** [25] is a scientific experiment that uses Internet-connected computers in the Search for Extraterrestrial Intelligence (SETI). You can participate by running a free program that downloads and analyses radio telescope data. With 3154517 users taking part there has been a total CPU time of 668852.233 years (as of 19th July 2001) and this figure is growing on a daily basis.
- **Gnutella** [19] is an open, decentralised, P2P search protocol that is mainly used to find files. By April 2000, one Gnutella implementation called Fast-Track [26] matched the popularity reached by Napster.
- **Napster** [27] is the controversial music file-swapping application, had reported 6.7 million users by August 2000. In February 2001, Jupiter Media Matrix [28], the global leader in market intelligence, reported that Napster was used by 14.3 percent of online users at home in thirteen leading wired countries. Note that Napster is not a true P2P system since the availability of peers is located through a central database at Napsters web site.

An important aspect also supported in these systems is support for peer discovery and routing. An important prob-

lem in the context of P2P systems has been the effect of the heavy traffic generated as a result of message interaction between peers – especially as there is no centralised control in such systems. A number of P2P application utilise a “flooding” mechanism to forward messages to maximise reachability. This severely restricts the scalability of such approaches – and various approaches to overcome this problem have been investigated, such as [7]. This issue is of particular importance in the context of a Consumer Grid – where a potentially very large number of resources (nodes) may participate. Currently, we utilise the discovery processes within JXTA [16] to achieve this – relying on Triana peers to be discovered based on very simple attributes – such as CPU capability and available free memory. We also aim to explore additional capabilities of a peer to support this discovery process – in particular the ability to group peers with common capability into virtual peer groups.

A security aspect not addressed yet is the actual application that will utilise the Consumer Grid – for instance, although a user may agree to contribute their resources (by hosting a Triana peer), they would not have direct control of what application actually utilises their resource (this is the classic problem of being unable to prevent a collection of computational peer resource being used to crack passwords). This is a difficult problem to overcome – as it is possible for a user to disguise the computational tasks they distribute to peers – and therefore difficult to detect. An alternative approach would be to allow users to only download executables that are selected from a pre-agreed, certified, software library. We believe that the Triana peer environment will help us investigate some of these issues – by providing a test bed for experimentation.

## 4 Conclusions

This paper describes the Triana software for developing applications using Grid and P2P infrastructure. Triana supports the creation of applications by migrating code to a resource where execution is desired. This is achieved by discovering peers that offer particular computational capability and then downloading the code to these peers for execution. The Consumer Grid idea is centred on the notion that individuals or organisations may wish to contribute computational resources, with resources possessing different computational capabilities and access patterns. Two ideas from P2P systems are utilised in this system: (1) File sharing ideas from systems such as Napster and Gnutella, whereby executable and data files are transferred to the point of execution. Peer naming, grouping, and advertising is achieved using JXTA, (2) sharing of (unused) computational cycles managed by the resource manager, from systems such as SETI@HOME, Entropia [29] and Parabon [30]. Triana provides a user interaction facility that makes the existence of

such an infrastructure transparent to the user. We believe that the success of P2P systems (such as Napster) and utilisation of Web servers to supporting both file sharing and CPU sharing, will be important to support a wider use of Grid technologies and applications.

## 5 Acknowledgements

Thanks to Dr B. S. Sathyaprakash for providing the technical specifications concerning the sampling rates, templates and execution estimations for the binary-star user scenario.

## References

- [1] Computing Industry Almanac. See Web site at: <http://www.commerce.net/research/stats/wwstats.html>
- [2] The Globus Project: <http://www.globus.org>
- [3] Programming the Grid: Distributed Software Components, P2P and Grid Web Services for Scientific Applications. Dennis Gannon, Randall Bramley, Geoffrey Fox, Shava Smallen, Al Rossi, Rachana Ananthkrishnan et al. Department of Computer Science, Indiana University, USA.
- [4] The XCAT Science Portal, S. Krishnan, R. Bramley, D. Gannon, M. Govindaraju, R. Indurkar, A. Slominski. Proceedings of SuperComputing, Denver, November 2001.
- [5] Gateway Computation Portal, Geoffrey Fox et al.: [http://www.computingportals.org/CPdoc/Gateway\\_CP.doc](http://www.computingportals.org/CPdoc/Gateway_CP.doc)
- [6] NPACI GridPort Toolkit and HotPage UCSD User Portal, Mary Thomas et al.: <http://www.computingportals.org/CPdoc/HotPage.doc>
- [7] F. S. Annexstein, K. A. Berman, M. A. Jovanovic, "Latency Effects on Reachability in Large-scale Peer-to-Peer Networks", Proceedings of SPAA, 13<sup>th</sup> ACM Symposium on Parallel Algorithms and Architectures, 2001
- [8] Java Sandbox: <http://java.sun.com/marketing/collateral/security>
- [9] Foster, I. And Kesselman. C. (eds) The Grid: Blueprint for a New Computing Infrastructure. Morgan Kaufmann, 1998.
- [10] Grid Information Services for Distributed Resource Sharing. K. Czajkowski, S. Fitzgerald, I. Foster, C. Kesselman, Proceedings of the Tenth IEEE International Symposium on High-Performance Distributed Computing (HPDC-10), IEEE Press, August 2001.
- [11] Foster, I. And Kesselman. C. The Globus Project: A Status Report. In Proc. Heterogeneous Computing Workshop. IEEE Press, 1998, 4-18.
- [12] I. Foster, C. Kesselman, J. Nick, S. Tuecke, "The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration", February, 2002. Technical Report, Argonne National Laboratory, Chicago, USA. Downloadable as: <http://www.globus.org/research/papers/ogsa.pdf>
- [13] The Global Grid Forum: <http://www.gridforum.org>
- [14] IBM Research, "Web Services Toolkit". See Web site at: <http://www.alphaworks.ibm.com/tech/webservicestoolkit/>
- [15] The Community Resource for Jini™ Technology: <http://www.jini.org/>
- [16] Project JXTA: <http://www.jxta.org/>
- [17] The Triana Software Environment: <http://www.triana.co.uk> and <http://www.trianacode.org>
- [18] GEO 600: <http://www.geo600uni-hannover.de>
- [19] Gnutella: <http://gnutella.wego.com/>
- [20] The Common Component Architecture (CCA): <http://www.acl.lanl.gov/cca/>
- [21] GridOneD: <http://www.gridoned.org/>
- [22] GridLab: <http://www.gridlab.org>
- [23] Inspiral Binary Search. See Web site at: <http://www.astro.cf.ac.uk/groups/relativity/research/part12>
- [24] Condor: <http://www.cs.wisc.edu/condor/>
- [25] SETI: <http://setiathome.ssl.berkeley.edu/>
- [26] FastTrack: [http://www.fasttrack.nu/index\\_int.html](http://www.fasttrack.nu/index_int.html)
- [27] Napster: <http://www.napster.com>
- [28] Jupiter Media Matrix: <http://www.jmm.com/>

[29] Entropia:  
<http://www.entropia.com/>

[30] Parabon:  
<http://www.parabon.com/>