General Purpose Grid Computing In a Peer-to-Peer Network

Austin Foxley Tim Prouty Jeremy Hammer

April 2nd, 2006

1 Operational Concepts

Currently there is no easy way to write a program that takes advantage of a distributed network of computers to perform a computational job. There are specialized systems available, such as SETI@home that searches for extraterrestrial intelligence, that do specific computations. There are also frameworks available that allow distributed networks of computers to be organized into grids for computation. We are proposing to take the idea of grid computing to the next level of generality and usability by combining the concepts of grid computing with Bit Torrent.

Bit Torrent is a peer-to-peer network technology that breaks files into many pieces so files can be downloaded efficiently from all of the individual computers (nodes) that are trying to download a particular file. Users can begin downloading a new file with Bit Torrent by joining the file's torrent. We are extending this idea of a torrent to grid computing by breaking up a computational job into smaller tasks and then allowing nodes in the grid to perform those tasks. For each job there is exactly one host and zero or more clients. The host is the node that originally created the job, and the clients are the nodes that are actually performing distributed tasks that the host needs done. To maintain generality, it is necessary for clients to run arbitrary untrusted code from the host. This potentially huge security hole is solved through the use of sand boxing, which allows untrusted code to run inside of a container that guarantees the code is run safely.

This new grid technology will make it much easier for a programmer to write a computationally intensive system that can take advantage of a massively parallel grid. Our proposed framework will allow the programmer to use standard multi-threaded programming idioms to take advantage of the grid. In addition, the Bit Torrent based tracker will make it easy for clients to join the grid and contribute to the computation. This system could be useful publicly on the Internet for jobs such as ray-tracing, encoding media, computing prime numbers, and even porting systems like SETI@Home to a more general framework. Privately, it could be used by businesses to perform internal computations on idle computers in their network, and also by governments for super computing.

The scope of this project involves creating the host program that allows a new grid-aware program to be written and run, the client program that actually runs the tasks created by the host, and a tracker server that keeps track of peers currently in the grid.

2 System Requirements

The essential feature of the system is creating a multi-threaded, shared memory environment that is distributed across a loosely coupled network of peers (grid). Features that are needed to implement this are:

- A mechanism for making the use of shared memory and multi-threading constructs such as locks (almost) transparent to each node.
- A mechanism for making the computation safe. This is need because the network is inherently untrusted and the end result of the system will be nodes running arbitrary code. This obviously needs to be made safe.
- A mechanism for determining and maintaining peers in the grid.

The deliverable of the system will be in several parts. There will be a web service known as the "tracker" which will maintain a list of all the peers in the grid. It is the responsibility of the tracker to keep the list current. When a peer wants to join the grid, it will call the web service to determine the current members of the grid. Once that is done, the tracker does not need to be involved in the evolving computation.

The next part of the deliverable will be the long running grid monitoring application. It will be started on system start up and will manage what trackers a node is connected to, and will be responsible for starting and distributing tasks. To do this it will interface with remote nodes and determine whether they are available for use. This component will be responsible for maintaining fairness metrics on a local level. Connected to this component will be possibly many local applications on the node that want to use the grid services.

The local applications component is simply an application that is written against a standard API for using the grid services. It can be any application that can be written in parallel that can run in the sand-boxed environment of the grid. An example might be a ray tracer, or a movie encoder.

The standard API component will be a library available for use applications for writing applications on the grid. It will be responsible for managing the shared memory and multi-threading aspects of the system. The interface to other nodes will be an RPC protocol for getting and sending serialized objects.

3 System and Software Architecture

There are two phases to using this system, development and deployment. During the development phase, the programmer must know he/she is developing an

application to be used on a distributed grid. The API given to the programmer will be a multi-threading API much like the API used in standard multi-threaded programming. The programmer does not however have to worry about how the job will be distributed within the grid.

Since we will not have compiler support, there will be special ways in which memory must be accessed. When memory is accessed normally, it reads or writes on the local machine. Since it will be running on a distributed system with shared memory, each read and write will have to make sure it has the most recently updated version for synchronization. Any object shared between multiple threads will have to inherit from a special class to allow for the network sharing.

Once an application has been developed it is ready to be deployed onto a distributed grid. To do this, the user will connect to a grid and start its job. Jobs are then distributed to other nodes on the grid. From a network point of view there are only three different kinds of nodes: tracker, host and client.

The purpose of a tracker is to keep track of who is connected to a particular grid. When a node connects to the grid, it registers itself with the tracker. As part of the grid connection process, the tracker sends the connecting node a list of IP addresses of other nodes connected to the grid. After connection, all communication is peer-to-peer.

The concept of host and client only occurs within the context of a particular job. A host is a node which instantiates the job and splits it into different tasks. The host then finds as many clients as it needs, or how many are available, and then distributes the tasks to the clients.

A possible way to find clients is to broadcast a request for clients and the currently idle clients respond. The host can then pick which ones of the clients which responded to use. This will handle busy clients and allows for statistical analysis when choosing clients.

4 Life-cycle Plan

The users of this project will be in two distinct categories: 1) users who write new jobs 2) users who are performing tasks for a job. The users that write new programs will generally have a very strong programming background. The users that are performing tasks for a job don't need to be as technically knowledgeable, but will still probably have a decent technical background. The other major stakeholders in this project will be the original architects and the development team. The roles and responsibilities of the stakeholders will primarily be in helping to refine how the overall system will be implemented to make it easy to use and still maintain its power and generality.

After some initial specifications of interfaces, much of this project can be completed in parallel. The most challenging component of this project will be creating the core API for the distributed grid system. A working version of this should be competed by week 4. At the same time, resources should also be allocated to writing the tracker server, and some test programs that can

run on the grid. Once these core components are working, enhancements for performance and additional features can be added. Some additional features would be: grid processing fairness, distributed garbage collection, etc.

More detailed scheduling and allocation of resources will be provided once the team is assembled for this project.

5 Feasibility Rationale

This project can be split into many different pieces. Network management, memory management, memory and task distribution, grid management, testing, etc. We feel given the resources indicated this will make a good 9-10 week project. There are however some risks involved.

Since a lot of the project is directly related to the core goal, there is not a lot of room for additional features. This directly affects our ability to cut features out if we get off schedule to have a release done on time.

Most of the technology is already in place for us. Network protocols such as TCP/IP give us a reliable base to work upon. Many languages today such as C# and Java provide methods of "Sand-boxing" and remote connections. Using these tools significantly simplifies this project and brings it into a manageable scope for the resources we have to work with.