Presented at the 2005 BoF LinuxForum in Copenhagen October 8th 2005

Tapping the Matrix: Revisited[1]

Carlos Justiniano
ChessBrain Project
http://www.chessbrain.net
cjus@chessbrain.net

Abstract

The software infrastructure used on Volunteered Distributed Computing is evolving to meet the diverse needs of researchers. These emerging systems are being built using Open Source tools. We describe a number of such systems in the context of this emerging field. Throughout this paper we revisit concepts presented in the April 2004 paper entitled, "Tapping the Matrix".

1. Machines are underutilized

Modern machines are capable of executing billions of instructions in the time it takes us to blink. This fact may be less surprising when we consider that the typical machine's sold today feature processors running at multiple gigahertz supported by hundreds of megabytes of main memory.

Surprisingly, the vast majority of personal computers are underutilized. The truth is many machines are idle for as much as 90% of an entire day. Even when active, most applications utilize fewer than 10% percent of the machines CPU.

Furthermore, this trend shows no signs of reversing, in fact, conservative estimates indicate that there are roughly 800 million personal computers in use. About 150 million are Internet connected machines which are expected to increase to one billion by 2015 [2]

It is an observable fact that the vast majority of the desktop computers found in businesses, universities and homes are largely idle. This author works for a company that has a work force of over 30,000 employees; most employees have one to two machines at their desk. Most machines are always on, largely because our IT department monitors machines and applies software updates as required. Additionally, many of our high-tech employees have powerful desktop machines at home which are used to connect to our offices via our virtual private network (VPN). While at work most of those machines are largely inactive. This situation isn't new.

2. Researchers take notice

During the mid-1990 the general public began discovering the world-wide-web. This was a time when early adopters began connecting machines to the Internet to retrieve email and explore the rapidly evolving web. Early researchers began to realize that the large number of machines connecting to the Internet could potentially be harnessed to form large virtual supercomputers.

In early January 1996, a project known as, The Great Internet Mersenne Prime Search (GIMPS) began an Internet distributed computing effort in search of prime numbers. The project focused on Mersenne numbers, numbers which are candidate prime numbers, but not necessarily prime. A month later the project reported the involvement of 40 people and about 50 computers. [3]

The following year, Earle Ady, Christopher Stach and Roman Gollent began developing software that would enable network servers to coordinate a large number of remote machines. An initial goal was to compete in RSA's 56-bit encryption challenge. Encryption schemes are susceptible to brute-force attacks given sufficiently powerful hardware. The project was going to need access to a large number of machines in order to tackle an encryption key space consisted of 72 quadrillion encryption keys. The group, which later became known as distributed.net, set out to harness the distributed computing power of remote machines to launch a massive encryption key attack.

By October the Distributed.net project discovered the correct key to unlock the RSA challenge message: "*The unknown message is: It's time to move to a longer key*

length". In October, the New York Times published an article on their achievement entitled, "Cracked Code Reveals Security Limits". [4]

While both GIMPS and Distributed.net's efforts achieved public recognition, another project would go on to become a household name in distributed computing circles. In 1998, a group of researchers at the University of California in Berkeley launched the SETI@home project. The project uses Internet-connected computers to aid in the search for extraterrestrial intelligence. SETI@home captured the public's interest and grew to several hundred thousand contributors.

SETI@home's appeal was fueled by the general public's interests in the possibility of extraterrestrial intelligence and by the then timely release of films such as The Arrival and Contact (which was based on Dr. Carl Sagan's book).

Another important reason for SETI@home's success is due to their use of the screensaver application format. During the early 1990s screensavers were quite popular, featuring flying toasters and mildly hypnotic multi-color swirling patterns. By the time SETI@home first appeared, the general public already understood that screensavers become active when machines were not in use. The perceived non-intrusiveness of screensavers eliminated the barrier of entry onto millions of desktop computers.

The project's success brought with it a considerable amount of computing power. In 1998 project leader and researcher, Dr. David Anderson compared, SETI@home's distributed computing potential to the fastest computer built at the time, IBM's ASCI White. IBM built the 106 ton, $110 Million dollar system for the U.S Department of Energy. The supercomputer was capable of an impressive peak performance of 12.3 TFLOPS. Anderson wrote that SETI@home was faster and cost less than one percent to operate. [5] Naturally, the cost savings are due to the fact that SETI@home computations are distributed and processed on remote machines which are paid for, operated and maintained by the general public.

GIMPS, Distributed.net and SETI@home are still active. GIMPS, the longest running distributed computing effort, recently discovered the existence of a prime number consisting of over seven million digits! Distributed.net has successfully completed a number of encryption challenges, and SETI@home has been instrumental in raising public awareness for distributed computing efforts. During the past few years the SETI@home group has created the Berkeley Open Infrastructure for Network Computing (BOINC) platform which is already helping to launch new distributed computing projects. We'll take a brief look at BOINC later in this paper.

3. Resources exist behind locked doors

Potential distributed computing resources can be found in millions of locations throughout the world. However, each location has at least one thing in common. People, people control access to these resources. To paraphrase the 1999 block buster movie, The Matrix: They are the gate keepers. They are guarding all the doors; they are holding all the keys.

Researchers are faced with the difficult challenge of creating projects which are of interest to sizable groups of people. When the public takes interest, the virtual doors leading to computing resources are unlocked and machines begin to connect from diverse environments.

Where might machines connect from? We'll examine a few general environments where volunteer computing projects can expect machines to connect from.

3.1 Individual machines

Millions of homes have computers which are capable of connecting to the Internet. Many homes contain more than one machine, often connected via a private home network. Furthermore, an increasing number of machines are connected via broadband, always on, connections. These trends are showing no signs of diminishing.

Figure 1. The author's home environment during early ChessBrain testing. A bit atypical of home environments however, many homes have machines in different rooms.

3.2 Garage farms and clusters

Some computing enthusiasts go as far as to build and operate their own clusters. The number of people who actually do this is larger than one might think! There exists an Internet sub-culture of enthusiasts who refer to themselves as DC'ers. They form and participate in distributed computing teams sometimes numbering in the thousands.

Figure 2. A ChessBrain contributor proudly displays his server farm.

3.3 Businesses and Universities

A multitude of machines exist in businesses and universities throughout the globe. These machines are often tightly secured and controlled. However, people control access to these machines and they often grant access by allowing their machines to run well behaved distributed computing software.

Figure 3. The 240 node BioCluster which participated during the ChessBrain.net Guinness World Record attempt in Copenhagen.

4. The Volunteer Computing model

The term “Distributed Computing” is well recognized by practitioners and participants alike; however, the term has lost its once specific meaning. The general media now uses the term “Distributed Computing” to describe web services and service oriented architecture (SOA) solutions. A newer term is necessary in order to differentiate the form of distributed computing which we’re considering in this paper from the now overused term.

Distributed computing has been referred to as “Grassroots Supercomputing” and “Public Computing” however those terms fall short of what people actually do when they contribute as part of a distributed computing project. The inescapable fact is that in order for distributed computing projects to work they must have volunteers. The term “Volunteer Computing” has emerged to describe distributed computing projects where volunteers supply the necessary computing resources.

4.1 How volunteer computing works

Most volunteer computing projects (with the exception of a small number) are implemented as client-server applications. End users are required to actually place software on their machines. This is what distinguishes a volunteer computing project where participation is “active” from the relatively passive use of web enabled services. The presence of an *active role* is why volunteer computing is often refereed to as Peer-to-Peer computing rather than *client-server* from a traditional web-centric view.

Furthermore, the vast majority of volunteer computing projects do not rely on the client-side existence of runtime frameworks such as Java or .NET (although this will likely change). This impacts project developers who are forced to implement native solutions for each platform they wish to support.

These important distinctions introduce additional levels of complexity which we’ll examine in greater detail.

4.1.1 A twenty thousand foot view

Fundamentally, users download and install a relatively small client application which is capable of communicating with project servers to retrieve individual work units. Each unit of work contains the

data and in some cases - the instructions that a client node can use to process the work.

Upon completion the client application sends the results of the work unit to a project server, which is responsible for collecting results and performing any post processing that may be required.

A project server may in turn use the processed results to determine how to generate newer work units for subsequent distribution.

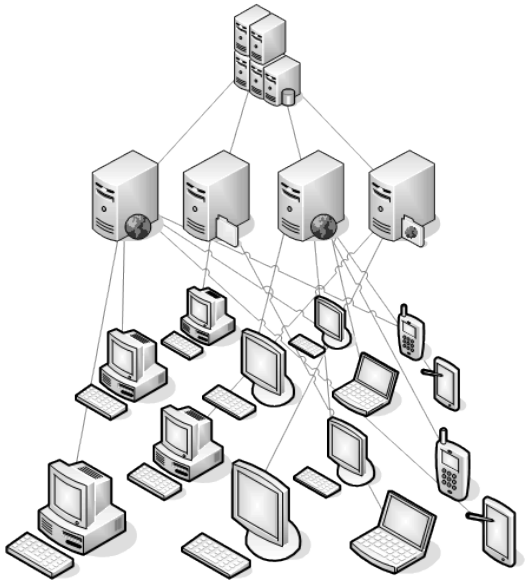

Figure 4. Client nodes communicate with web and file servers which in turn communicate with backend application and database servers.

4.1.3 Client-side considerations

When an end user agrees to run a piece of software on his / her machine it is because at some level the user trusts that the software is implicitly safe and well behaved. We explored issues related to trust in our original paper entitled "Tapping the Matrix" [1]

A significant obstacle for volunteer computing project developers is the creation of native client-side applications. Not only must the software perform the primary task (the remote computation) it must do so without requiring additional preinstalled client-side software. It's important to streamline the end user experience of acquiring and operating the project software – otherwise a usability barrier will be created. If the software is too difficult to setup – users will leave to find other projects to participate in.

The challenge for project developers is to create applications which are relatively self contained (or include all the files it will need) in an easy to download and install package. Additionally, developers must choose which native platforms to support. Because volunteer computing projects typically require an extensive number of participants the need to support the Microsoft Windows platform is somewhat inescapable. Most project developers choose to support MS Windows, GNU/Linux and Apple OSX – in that order.

Further complications become apparent when the developer realizes that the client-side application must include the ability to navigate firewalls and proxy servers. Additionally, communication must be secured using encryption in order to ensure that the information sent to the project servers has not been tapered with.

In contrast, project developers have a great deal of control over the choice of backend solutions. The introduction of the client-side component presents a significant hurtle. Fortunately, comprehensive solutions have emerged in recent years.

5. Open Source Tools

Open Source tools are playing a vital role in the development of volunteer computing projects. The decreasing costs of commodity hardware, coupled with the free availability of highly capable software, have made it not only possible, but also economically feasible. Economics however is only one of many reasons. The strength of the open source community has given intrepid researchers the sense that they are far from alone in their efforts.

During the development of the ChessBrain project we encountered many open source tools. The tools that project developers choose depend largely on their own skill sets and personal preferences. In this paper we restrict our investigation to solutions which do not require a client-side runtime framework. However, for developers who don't consider this an issue we invite them to consider the use of Java/JXTA/P2P Sockets and Mono .NET.

At the end of the next section we describe our own choices and rationales.

5.1 LAMP Building Blocks

The GNU/Linux, Apache, MySQL and P-scripting language tools form the building blocks for a great many open source projects. While GNU/Linux itself is predominantly represented in the LAMP acronym, it is in fact replaceable by other fine operating systems.

The use of Apache, MySQL and a scripting language such as PHP, Perl or Python enable developers to create web accessible services. Because most volunteer computing projects are client/server applications, the LAMP toolset is ideal for developers who wish to construct their own solutions – but don't want or need to build the underlying software infrastructure.

The challenge in using LAMP based tools is that the project developer must still consider how to tackle the client-side application.

One approach, which we're adopting for ChessBrain, is to use a cross-platform development tool such as WxWidgets and GNU g++ which simplifies the creation of Windows, Linux and Mac GUI applications. [4] The solutions we examine next embrace LAMP tools to varying degrees.

5.2 Jabber

Jabber was first developed in 1998 by Jeremie Miller as an open source Instant Messaging system and viable alternative to propriety IM solutions such as the AOL, ICQ, MSN and Yahoo messengers. [7] Over the years the underlying Jabber protocol was formalized into the Extensible Messaging and Presence Protocol (XMPP). Today, XMPP is endorsed by IBM, Google and many others.

Jabber quickly outgrew its humble IM beginnings to embrace XML based messaging. Because, volunteer computing is largely concerned with message based communication between servers and clients, the use of a messaging server is quite natural.

The Jabber community has developed both server and client-side software which is freely available. The existing code base offers project developers a significant advancement in creating their own volunteer computing solutions based on Jabber.

XMPP is a secure XML streaming protocol which addresses many of the issues we've raised earlier in this paper. [8]

The only disadvantage with Jabber which we've identified is the dependency on XML and TCP. However, when XML is properly utilized the disadvantage may become slight and even insignificant.

5.3 GPU

The Global Processing Unit (GPU) project is a framework for distributed computing over the Gnutella P2P network. [9]

GPU developers decided to build their framework using the propriety Borland Delphi rapid application development environment (featuring a Pascal dialect). This presents an obstacle for developers who don't know Pascal or haven't looked at the language recently.

In addition, GPU relies on the Gnutella P2P network for communication services such as peer and resource discovery. A consequence of this decision is that many businesses and universities block Gnutella network traffic, whereby limiting the number of potential project contributors.

GPU's founder Tiziano Mengotti also points out that the use of Gnutella limits the available pool of machines to about 2000. This can be a severe limitation for projects requiring thousands of machines for effective problem decomposition and distribution.

We believe GPU is an intriguing project with obstacles that may be overcome with some effort. The project is actively exploring P2P grid concepts and is certainly worth a look.

5.4 BOINC

For developers unable or unwilling to invest in creating custom solutions there is the Berkeley Open Infrastructure for Network Computing (BOINC) project. BOINC was created by the SETI@home group. Today, the framework is used by a growing number of high profile projects. [10]

BOINC project contributors led by Dr. David P. Anderson (with the Space Sciences Laboratory at the University of California at Berkley); have leveraged their experience on SETI@home to create a generalized solution intended to meet the needs of most project developers. Consider the following benefits:

- Open Source. Both client and server software is freely available.
- Secure client/server communication.
- Built on top of LAMP tools.
- Well defined application programming interface for developers of both client and backend server applications.
- End users can chose to participate on one or more volunteer computing projects.
- Name brand recognition of having your project recognized and loosely associated with both BOINC and other high-profile projects.

BOINC is a solution that is ready for use today. The solution is well documented and well worth deeper considerations.

5.4.1 BOINC at a 20 thousand foot view

End users download a one to six megabyte file (size depending on their target platform and project specific additions.) onto their machine in order to participate on one or more volunteer computing projects.

On the backend server-side BOINC consists of several ready-made server components which communicate with one another and a MySQL database server. The Apache web server is used with FastCGI and Python to create web interfaces.

5.4.2 BOINC: a closer view

As a BOINC project developer one is only responsible for creating the client-side and server-side behavior that addresses a specific project's needs. The BOINC framework contains clearly designated areas where one is responsible for adding project specific code modules.

The development platform consists of GNU C++ on the server side, and GNU C++ or Microsoft Visual Studio / .NET on the client side.

The server environment is expected to be a Linux box with MySQL, Apache and Python installed.

The BOINC server side solution utilizes the MySQL database server to store and retrieve project specific information such as work units, results and user account information.

The diagram in figure 5 shows a user interacting with the BOINC system. The client side components are shown to reside in the user's machine. Below the client-server separator line we see the components of the BOINC backend. The two bold boxes represent the location of client and server project specific modules.

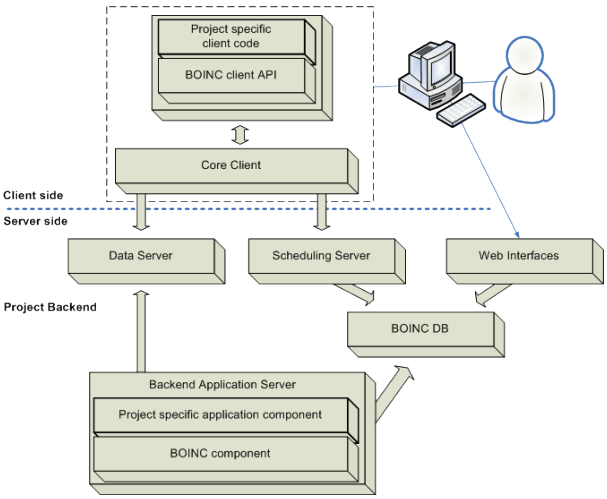

Figure 5. The BOINC architecture.

5.4.3 BOINC disadvantages

We're compelled to state that BOINC's advantages will very likely outweigh the disadvantages we'll list in this section. The information is provided for brief consideration as one needs to realize that BOINC is evolving to meet emerging needs. Visit the project website to learn of new developments.

During the development of the new ChessBrain II project we identified the following issues while investigating BOINC.

- Despite BOINC's tool support, project developers need to understand and feel comfortable with a number of technologies in order to gain control over their project. In short, BOINC doesn't yet offer out-of-the box supercomputing – although, at this time it comes closer than any non-commercial tool in existence.
- Complete reliance on LAMP excludes MS Windows environments on the server-side.
- The BOINC project is structured in the traditional view of client / server architectures. We feel that a P2P view is the emerging future of volunteer computing efforts.

- Uses XML over HTTP which may result in larger communication packets than some project developers may desire.
- End user machines are seen as compute nodes. We feel that volunteer computing projects will need to increasingly embrace server farms, Beowulf clusters and Grid systems.

According to project leader Dr. Anderson, BOINC was specifically created for scientists – not necessarily software developers and IT professionals. The goal of BOINC is to enable scientists to easily create volunteer computing projects to meet their growing needs for computational resources. Toward this end, BOINC is a remarkable achievement which will continue to have a profound impact on the future of scientific research.

6. ChessBrain II and msgCourier

ChessBrain is a volunteer computing project which is able to play the game of chess against a human or autonomous opponent while using the processing capabilities of thousands of remote machines.

ChessBrain made its public debut during a World Record attempt in Copenhagen in January 2004, during a live game against top Danish Chess Grandmaster, Peter Heine Nielsen.

What makes ChessBrain unique is that unlike other volunteer-based computing projects, ChessBrain must receive results in real time. Failure to receive sufficient results within a specified time will result in weaker play. Tournament games are played using digital chess clocks where the time allotted per game is preset and not renegotiable. So while ChessBrain waits for results its clock is counting down.

Our goal on ChessBrain has been to create a massively distributed virtual supercomputer which uses the game of chess to demonstrate speed-critical distributed computation. We chose chess because of the parallelizable nature of its game tree analysis, and because of our love for the game. [11] This makes the project professionally rewarding as well as personally enjoyable.

The existing design of ChessBrain features a single Intel P4 3.0 GHz machine. The machine hosts a database, the ChessBrain SuperNode server and a chess game server. During the exhibition game against Chess Grandmaster Nielsen, the machine was overloaded as it tried to support thousands of remote PeerNode clients. Prior to the event we speculated that perhaps a thousand machines might support the chess match. We were not expecting thousands of machines. Fortunately, ChessBrain made it through the match securing a draw on move 34.

We've learned a great deal during and after the event. Our improved understanding is being applied on ChessBrain II.

Perhaps the single biggest change with ChessBrain II is that we're replacing the idea of a single SuperNode with the concept of clusters of SuperNodes. Our vision is to see SuperNodes establish P2P relationships among collaborating SuperNodes. The software for each SuperNode is being designed to create communities of machines, where each community can consist of several thousand PeerNodes.

During the game against Grandmaster Nielsen several clusters consisting of over 200 machines, and a few others consisting of 50-100 machines, participated during the event. At the time ChessBrain wasn't designed to take advantage of clusters – like current volunteer computing projects, ChessBrain was designed for use on individual machines.

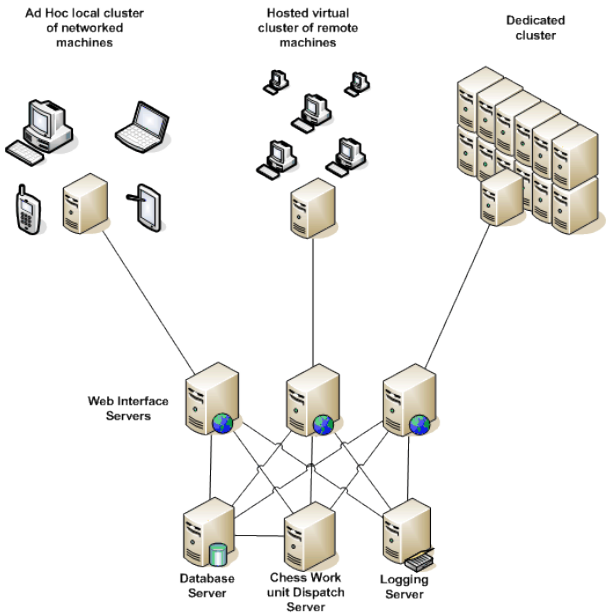

Figure 6. ChessBrain II.

Over time it became clear that ChessBrain software would have to embrace a wider spectrum of computing environments which include Beowulf clusters, compute farms and Grids.

Our vision for ChessBrain has evolved toward P2P distributed clusters, where volunteer computing enthusiasts are promoted to virtual cluster operators. In the parlance of graph theory, our goal has become to create more hubs. [12]

6.1 ChessBrain II powered by msgCourier

Promoting volunteer computing enthusiasts to virtual cluster operators is non-trivial. If we opted for a BOINC-like framework we would have to expect that each cluster operator would be knowledgeable in the use of open source tools such as MySQL and Apache.

This simply isn't practical. On the ChessBrain project our Microsoft Windows based contributors outnumber our Linux contributors two to one.

We quickly realized that our new SuperNode software would have to be a self contained product which can cluster local or remote machines with ease.

The new SuperNode software is being built on top of msgCourier in order to address the challenges we're facing on ChessBrain II.

The msgCourier is hybrid application consisting of built-in components that enable P2P messaging for distributed computing applications.

The easiest way to envision msgCourier is to think of it as a cross between a messaging queuing server, web server, and application server.

The msgCourier is being developed as an open source project which isn't only intended to support next generation volunteer computing but also a host of other potential applications. In the figure 6, each server box with connecting lines will run a msgCourier server.

6.2 msgCourier Technical Overview

A primary goal on msgCourier is to eliminate external run-time dependencies. We chose to build msgCourier using the C++ programming language with initial support for GNU/Linux and MS Windows. We leverage a number of free and open source components to create a robust server application.

In support of C++ we use the Boost programming library. We added scripting support to msgCourier by embedding the Tcl language interpreter. We use Maciej Sobczak's open source C++/Tcl library interoperability between C++ and Tcl. For our database needs we've embedded the SQLite SQL engine into msgCourier.

We're securing msgCourier communication using the Crypto++ library. For our network monitoring we're using the GraphViz graph visualization software.

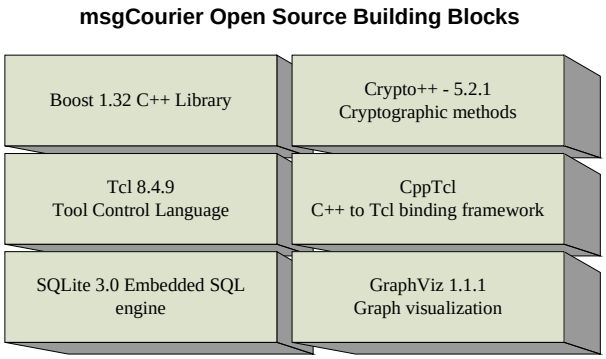

Figure 7. msgCourier Software Building Blocks

When using these components msgCourier is under two megabytes in size under MS Windows and about three megabytes under Linux. The cost of using these libraries really amounts to a more complex build process, but eliminates runtime dependencies.

We built msgCourier as a multithreaded application with support for both TCP and UDP based messaging.

Messages inside of msgCourier are handled by loadable components called message handlers. The msgCourier server can be configured to route messages to specific handlers on a local machine or another remote server. Figure 8 shows a list of components present in the current msgCourier application. The Crypto usage box on the bottom right is only currently partially implemented.

The msgCourier server has a built-in multi-threaded multiple connection web server component. Application developers can leverage the built-in web server to create their own custom configuration and monitor pages or web based services.

The use of HTTP and XML over TCP is optional. However, msgCourier has internal support for HTTP and XML because of their widespread ubiquity and ability to flow through firewall and proxy servers.

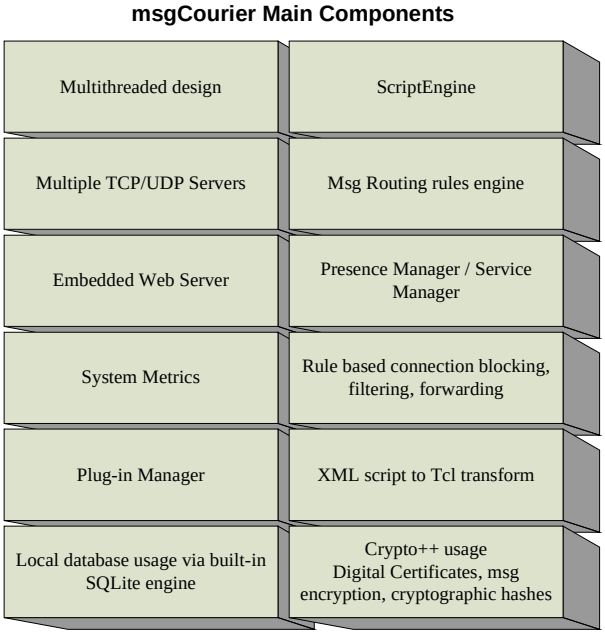

Figure 8. msgCourier Components

Some of our design choices were:

- No dependency on external software such as MySQL, Apache or external scripting languages. Although, project developers are free to embrace the technologies of their choosing.
- Self-contained. Uses embedded SQLite database engine and built-in Tcl interpreter.
- Configurable via a web based interface. Similar to Internet appliances.
- Based on a messaging paradigm similar to Jabber.
- Supports low-level communication via UDP for use within clusters.
- Uses its own optimized messaging format, but supports HTTP over TCP. XML is optional.
- Built-in (optional) XML to Tcl transform engine.
- Built-in web application server.
- Built-in message routing. Complex routing rules can use regular expression.
- Servers communicate with one another using P2P concepts.
- Plug-in component architecture allows developers to extend and customize msgCourier for their own specific needs.

6.3 msgCourier Project status

msgCourier is currently undergoing its first public testing. The framework for many of the features we've described is already in place and quickly evolving.

7. Emerging trends

We see the volunteer computing landscape evolving in sophistication to meet emerging project needs. A client-server centric view will become increasingly less common as P2P based solutions become the norm. Grid practitioners are already realizing that volunteer computing makes sense for certain aspects of their work. In addition they're discovering that the economic benefits of leveraging volunteer computing are well worth investigating.

Consider the frequency of which in-house clusters can be upgraded. Compare that to the upgrade patterns and increasing growth and availability of Internet connected machines in the possession of the general public. The exponential growth of the public computing sector is becoming difficult to ignore.

7.1 Embracing Clusters

Clusters exist throughout the world. Increasingly, clusters are being built and hosted in the homes of computer enthusiasts. Volunteer computing projects which specifically target clusters have an opportunity to benefit from the computing potential inherent in tightly coupled networked systems.

In this approach a node on a local cluster is designated as receiving batches of work units and or instructions for generating local work units from a project server. This local master node is then responsible for distributing work to local nodes whereby leveraging the benefits of local high speed intercommunication. Completed work units are then collected by the master node and sent back to the project server. This approach eliminates the need for each node in a cluster to individually communicate with a central project server. Thus, this solution makes effective use of the cluster.

7.2 Extending Grids

An increasing trend has become to explore how volunteer computing can allow Grid systems to freely tap the resources available in modern computing households.

It's important to realize that Grid systems while amazingly flexible are still relatively finite in capacity. In contrast, the availability of volunteer computing

resources increases based on public interests. A volunteer computing project that captures the imagination and interest of the general public can scale to an impressive amount of computing power in a matter of days.

We believe that Grid practitioners will find it increasingly compelling to explore extending their Grid facilities using volunteered computing resources.

7.3 P2P Clusters

In the earlier days of Internet distributed computation efforts, researchers realized that the Internet was quickly expanding and making it feasible to leverage remote resources. Today the following factors are helping to change our view of how future projects might be structured.

- Network speeds are increasing.
- The sophistication and interests of computer enthusiasts is expanding. Many are networking machines and exploring volunteer computing projects.
- The number of Internet connected machines continues to grow at impressive increments.
- Commodity hardware continues to increase in speed and storage capacities.

We're seeing that the sophistication of current computer enthusiasts has reached a point where it has become feasible to consider them in the role of remote cluster operators. The goal here is to distribute the bandwidth and processing loads to remote hubs. In addition, other benefits such as fault tolerance may be realized.

Volunteer computing software which is capable of P2P capabilities such as self organization will become increasingly common.

The vision we share with other practitioners is one of a P2P network of hubs which magnify the network potential of volunteer computing.

8. Conclusion

In this paper we've explored the field of volunteer computing. Along the way we've identified the vital role that the open source community is playing.

In conclusion we've shared our own recent developments and concluded with a look at emerging trends.

We invite you to explore this exciting field, which is still in its very early stages of development.

9. References

[1] C. Justiniano 2004. Tapping the Matrix. O'Reilly OpenP2P. http://www.chessbrain.net/documentation.html

[2] D.P. Anderson. Public Computing: Reconnecting People to Science, Nov. 2003. http://boinc.berkeley.edu/papers.php

[3] GIMPS: The Great Internet Mersenne Prime Search http://www.mersenne.org/

[4] C. Justiniano & C.M. Frayn 2003. ICGA Journal. The ChessBrain Project: A Global Effort To Build The World's Largest Chess Supercomputer. http://www.chessbrain.net/documentation.html

[5] Oram, A. (ed.) 1998. Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology. O'Reilly & Associates, Inc.

[6] WxWidgets: an open source, cross-platform native UI framework http://www.wxwindows.org/

[7] Jabber Software Foundation http://www.jabber.org/

[8] XMPP: Extensible Messaging and Presence Protocol http://www.xmpp.org/

[9] GPU: a Global Processing Unit http://sourceforge.net/projects/gpu

[10] Berkeley Open Infrastructure for Network Computing (BOINC). http://boinc.berkeley.edu/

[11] C.M Frayn & C. Justiniano 2004, International Conference on Scientific & Engineering Computation (Proceedings), The ChessBrain Project – Massively Distributed Inhomogeneous Speed-Critical Computation.
http://www.chessbrain.net/documentation.html

[12] Barabasi A. Laszlo 2002. Linked: The New Science of Networks