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

Early experiences with clusters and compute farms in ChessBrain II

Kevin Lew (rawr@inorbit.com)
Carlos Justiniano (cjus@chessbrain.net)
Colin M. Frayn (C.M.Frayn@cs.bham.ac.uk)

Abstract

Next generation volunteer-based distributed computing projects are working to embrace a wide range of distributed computing environments. In this paper we report on our early experiences with the ChessBrain II project, an established collaboration between researchers in a number of countries, investigating the feasibility of inhomogeneous speed-critical distributed computation.

1. Volunteer-based distributed computing

The term "Volunteer Computing" has emerged to describe distributed computing projects where volunteers from the general public supply the necessary computing resources.

Popular projects have successfully utilized thousands of distributed computers to tackle a vast range of highly complex, but separable problems. One such project is ChessBrain, a worldwide distributed network of machines that plays the game of chess.

2. ChessBrain and ChessBrain II

The ChessBrain project was formed in 2001 to study distributed computing and speed-critical computation using publicly volunteered resources. The game of chess was chosen because of the parallelizable nature of chess game tree analysis and because of the author's interests in the subject.

ChessBrain played its first game of chess using distributed machines in December 2002. On January 30th 2004, ChessBrain played a live game against top Danish Chess Grandmaster, Peter Heine Nielsen using 2070 machines from over 50 different countries. It was the first time in computing history that a distributed network of machines played a game against a top human opponent under tournament conditions. The game resulted in a draw after 34 moves and ChessBrain was awarded a 2005 Guinness World Record under the Internet section of the Science and Technology division.

ChessBrain is unique among volunteer computing projects because it requires real-time support from distributed machines. We are not aware of any other volunteer computing project which requires real-time feedback. The standard response timeframe is usually measured in days or weeks rather than seconds and minutes. The extra complexity required in order to synchronize and optimize the computational resources in order to cope with this time-criticality make distributed chess a fascinating area worthy of considerable study.

ChessBrain's real-time requirement has driven many of the architectural decisions that were made throughout the project's four year history. Many problems were encountered during the 2004 exhibition game against Grandmaster Nielsen. The ChessBrain team has embarked on a significant rewrite of the underlying framework in support of a faster, more robust and scalable approach.

The original ChessBrain system consisted of a central SuperNode server hosted at distributedchess.net. Thousands of PeerNode clients connected directly to the SuperNode to retrieve work units and to return them in real-time.

A single high-powered server is typically adequate for a small project. However, once ChessBrain grew to thousands of contributors a larger more scalable solution was required.

To address issues of both performance and scalability, a new design was chosen, consisting of clusters of SuperNodes. In this hierarchical model, each SuperNode maintains its own community of PeerNodes which communicate directly with it, and not with the central ChessBrain server.

This approach offers a number of tangible benefits:

- Internet latency is reduced for machines clustered around a SuperNode hub. In many cases these machines are located in the same building or at the very least in the same geographical region.

- Bandwidth requirements are greatly reduced as the bandwidth load is distributed among participating SuperNodes.
- Communication efficiencies are realized when machines exists within a local LAN environment and within clustered environments such as Beowulf clusters and compute farms – the so called Network of Workstations (NOW) and Pile of PCs (POP) arrangements.

During the past few years, we have received offers from computing centers which feature high speed clusters to host ChessBrain software. However, the first ChessBrain was not designed specifically for clusters so there has been very little compelling reason to commit a cluster to the project. During our demonstration match at Copenhagen in 2004, we were able to use the 200 node BioCluster located in Copenhagen, but due to the nature of the communications bottlenecks, we were unable to take proper advantage of the computing power it provided. We are addressing this limitation with ChessBrain II, which is specifically designed to harness distributed clusters. In so doing, we hope to localize messaging interaction between a SuperNode and its associated compute nodes whenever possible.

There is another significant benefit which is beyond the scope of this paper, which we'll touch on briefly. We believe that computer enthusiasts would find it entertaining to operate their own virtual supercomputers. Specialized software that easily allows enthusiasts to cluster both local and distributed machines would appeal to an Internet subculture of enthusiasts who refer to themselves as DC'ers (Distributed Computing practitioners). Because many DC groups contain hundreds of members (and many operate more than one machine), the social aspect of such an arrangement should not be overlooked.

3. msgCourier

A message is a fundamental unit of information exchange between distributed resources. All volunteer computing projects share the concept of message passing.

On ChessBrain II, it became clear that a more robust messaging infrastructure was required. Over a year ago, we began work on msgCourier - an open source custom server for use in P2P and non-P2P distributed computing projects. MsgCourier is designed to support distributed computing efforts where some combination of the following constraints are present:

- When it is undesirable to host multiple server applications, such as a web server, message queuing server, database server, and or an application server.
- When a server solution must support built-in security features which are comparable to but without requiring the use of SSL or SSH.
- When the server application must provide built-in Peer-to-Peer functionality such as, resource and service discovery, self-organization, and clustering.
- When the end-user isn't knowledgeable enough in the use or configuration of the applications listed in this section.
- When there is a need for a light-weight, high-speed server application base for use within embedded, resource constrained, and or low-cost hardware platforms.
- When the experience base of the application developer is limited with regard to network programming and thus a simple framework is required to build larger more complex applications.
- When ease of deployment and near-zero configurations are critically important.
- When the size of a completed application, which includes at least some of the functionality listed above, is important.
- When a desired solution must run under Microsoft Windows and GNU/Linux environments and must be portable to other platforms.
- When a non-propriety solution with source code availability is essential to ensuring maximum flexibility.

Development on msgCourier is being driven by the ChessBrain II project but we believe it will be of generalized use outside of ChessBrain. For example, we believe that the following applications are possible:

- Web servers, File servers, Proxy servers, Gateway servers, Load Balancing servers, Message/Content routers.
- P2P applications for resource sharing such as computation sharing, file sharing and storage.
- Game servers and chat servers that complement existing web based applications.
- Messaging Servers for Instant Messaging.
- Batch job processing systems
- Distributed computing solutions involving clusters.
- Solutions which wouldn't necessarily require the features of a Beowulf cluster or Grid solution, but do require a way to distributed work load.
- A queue/dispatch/director for work within an organized or ad-hoc compute farm.

It is important that we point out msgCourier is not intended to be a replacement for specialized web and messaging servers. Rather, msgCourier is intended as a potential stand-in component when the use of such systems is not feasible.

While considering our needs for ChessBrain II, we researched a number of potential solutions including the Berkeley Open Infrastructure for Network Computing (BOINC), but have concluded that ChessBrain's unique requirements necessitate the construction of a new underlying server technology.

The single most important reason why we've arrived at this conclusion is because our plans for ChessBrain II promote our end users to cluster operators. We believe that our user base consists of knowledgeable computer enthusiasts. However, two important factors limit the extent of their support:

- Many are computer savvy but are not networking specialists.
- Many don't have the time to invest in acquiring and configuring difficult to support and administer applications such as web, database servers and messaging servers.

The primary task for ChessBrain II is to minimize the complexity of clustering local and remote machines for our community of supporters. Our msgCourier server platform is intended to support many of our core needs.

4. Beowulf clusters and compute farms

With msgCourier, ChessBrain is able to communicate among a wide range of computing environments. In order to explore each environment we assembled a Beowulf cluster and compute farm, and have explored extending msgCourier with support for each unique environment.

4.1. Early Beowulf experimentation

Beowulf clusters are highly scalable clusters of commodity workstations running an open source software infrastructure.

We built a small Beowulf cluster consisting of three nodes and later expanded it to eight nodes. Owing to its unattractive appearance we affectionately refer to the cluster as the "Warthog."

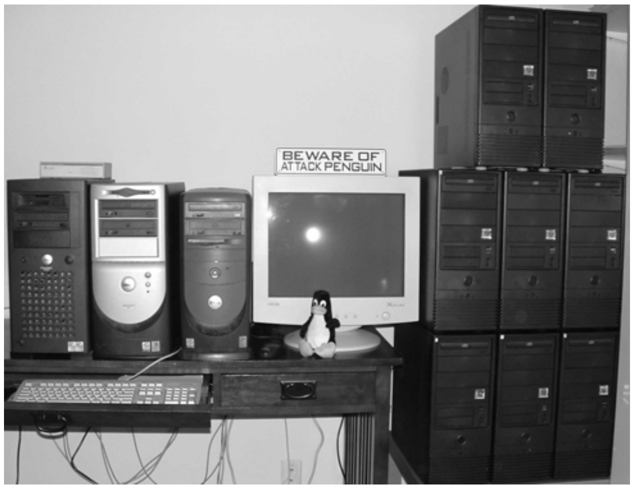

Figure 1. The "Warthog" cluster.

The three leftmost machines are the original nodes. Today, they currently serve as an NFS server, performance monitor and master node respectively. On the right are the eight new nodes. Each node is a basic machine, with the following hardware configuration:

- Motherboard:
  PCChips M863PRO3400A+ with a 333 MHz FSB.
- Processor:
  One Athlon XP-M 2600+ running at 1.8 GHz, soldered straight onto the motherboard.
- RAM:
  One 512 MB PC3200 400MHz DDR.
- Disk:
  One Samsung 40GB PATA drive @ 7200 RPM with 2MB cache.
- Connectivity:
  10/100 Mbps integrated (onboard) Ethernet.

The following diagram illustrates the overall network connectivity:

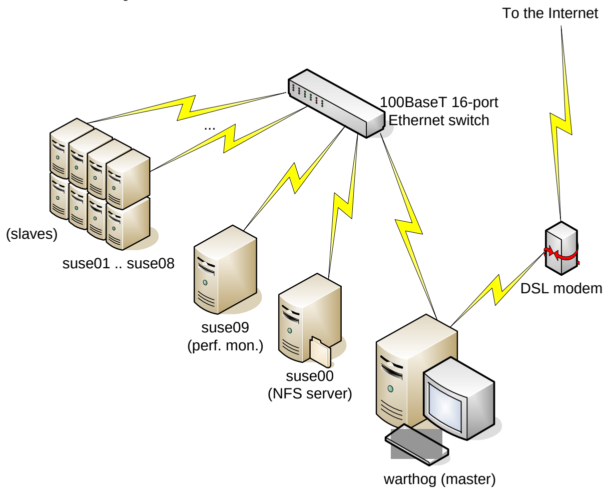

Figure 2. "Warthog" cluster connectivity.

Each node connects directly into a 16-port 10/100 Mbps switch. The switch supports the isolated subnet that is typical of Beowulf environments. In addition, the master node has a secondary Ethernet card connecting it directly to the outside world via a DSL modem. The master is also different in that it features a Pentium 4 processor running at 2.4 GHz and 2 x 512GB DDR memory. The NFS server is a dual-processor Pentium 3 with SCSI disks.

Much of the knowledge on how to build generic Beowulf clusters can be found in a variety of sources. We relied primarily on [Swendson 2004], with the following distinguishing features/differences:

1. We opted for SUSE Linux [OpenSUSE 2005]. At the time of writing, the Warthog runs SUSE Linux release 9.3. Our choice was influenced by previous experience with the distribution. This proved to be a good decision in light of the fact that OpenSUSE Linux installed smoothly and commissioning the nodes proved to be straightforward.
2. MPICH [MPI 2005] is the message passing API we chose for experimentation.
3. The master node doesn't run a firewall; however the DSL modem has an embedded firewall. The modem only has one port opened—the one through which ChessBrain interaction occurs with the outside world.
4. Our NFS server is not the master node. The rationale behind this is to restrict the NFS server's responsibilities.
5. Our slaves run a full OpenSUSE distribution with all the default packages; not just the network service package.
6. We use GRUB as boot loader, and we have it MBR-resident.
7. The NFS server boots first. Then the other nodes do so. When they do, they mount /mnt/beouser/ from the NFS server. beouser is the unix username under which we ran the MPI tests. The path /mnt/beouser/ is just a directory on the NFS server that holds executable files needed by the slaves. Every slave mounts /mnt/beouser upon startup.

4.2. Beowulf test application

The ChessBrain engine is fundamentally a sophisticated search engine. Intrinsically therefore, it is compute-intensive and we sought to implement a test-bed that would give us insight into the Beowulf's compute-intensive capabilities.

To that effect, we implemented a distributed prime-number search MPI program that uses brute force to determine primality. We plan to implement the same algorithm using msgCourier, and learn from the comparative outcome.

Our goal is not to make a case for choosing between an MPI and msgCourier. We are interested in both the issues and the implications of choosing a particular approach based on ChessBrain's needs.

The MPI implementation runs with a master node and a minimum of 1 slave. We experimented with up to 8 slaves. The testing proceeded as follows, using the simple test application of finding all prime numbers within a given range {1 … X}:

1. The master starts by finding out how many slaves are in the node pool.
2. The master apportions the integers by sending each slave a range delimited by a pair of integers, specifying the lower and upper bounds. For example if X = 84000 and there is 1 slave then that slave receives the pair (1, 84000). With X = 504000 and 2 slaves, the pairs are (1, 252000) and (252001, 504000).
3. Once the work is apportioned, the master waits to receive results from any slave.
4. Upon receiving their work unit from the master, each slave loops through its range and, for each odd number in the range, it uses brute force to find out if that number has factors other than 1 and itself. If it doesn't, then it is a prime.
5. As each slave loops through its range, it stores every prime number it finds in an array. Upon completing its work unit, the slave makes an MPI_Send call to report back its results.
6. The master receives arrays of prime numbers from all the slaves and collects them all in an array and eventually prints them out.

We used X = 84000, 168000, 252000, 336000, 420000 and 504000, and we harnessed anywhere from 1 to 8 slaves, distributing the larger integer ranges first, to allow their slaves to start first. Raw results in seconds are tabled below:

	Number of slaves							
X	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**
84000	5	4	3	2	1	1	1	1
168000	18	13	10	7	6	5	5	4
252000	38	28	20	17	14	11	10	8
336000	67	49	36	29	23	19	18	16
420000	101	75	55	43	36	30	26	23
504000	144	106	78	61	51	43	37	32

Figure 3. Raw performance results.

The chart below illustrates the timing results obtained. The six lines correspond to the different values of X used. The Y-axis represents the time taken to complete the search and the X-axis represents the number of slaves involved.

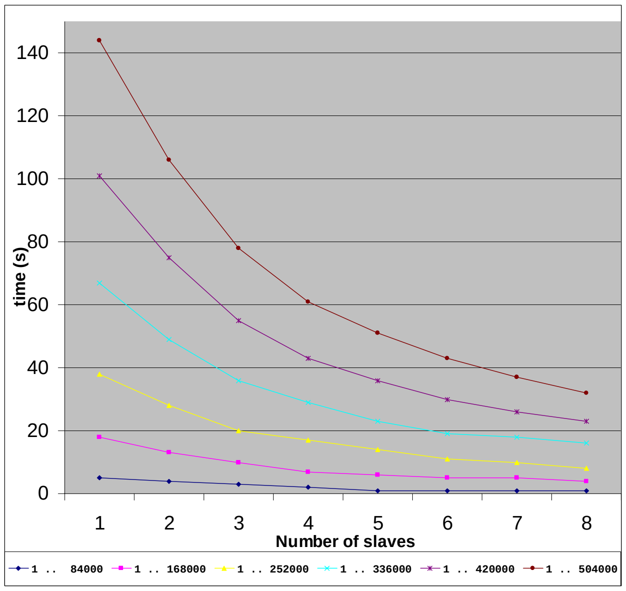

Figure 4. Performance chart.

The method by which work was divided up among the slaves led to uneven loads. For example with X = 504000 and 8 slaves, the task of slave 1 is to determine the primality of numbers 1 .. 63000 whilst slave 8 has to reckon with the range 63001 .. 504000. Slave 1 is bound to finish faster because the task of determining primality, at least in this naïve algorithm, is related to the size of the number to be tested. We did observe such an imbalance in the tests, noting that real-life frequently distributes uneven loads to nodes. Even in the first ChessBrain experiment, different compute nodes searched trees of varying depths.

Given this caveat, we examined scenarios where some of the larger integers were distributed to all slaves. For example, with X = 504000, we experimented with 80 MPI processes on 8 slaves, dividing the integers into 80 ranges. Again we distributed the larger integer ranges first, thereby allowing those processes to start first. We observed that the time taken for the Warthog to complete 1 .. 504000 dropped to 19 seconds.

The time taken for one slave to determine the primality of integers 1 .. 504000 was 144 seconds whereas, with 80 MPI processes on eight slaves, the time taken was 19 seconds. Theoretically therefore, were we to serialize the eight slaves, the task would take 152 seconds. This prompts us to posit that the Warthog loses some 5.6% compute efficiency which we attribute to MPI communication overhead. Those figures are heartening and we recognize that the compute-intensive and easily-parallelizable nature of primality determination meant that our tests leant themselves to good results. ChessBrain displays similar properties and we anticipate great gain from cluster-based parallelization.

An area msgCourier differs from our prime number search, primarily in the data type transmission between master and slaves. MPI requires the use of predetermined data types, while msgCourier isn't concerned with that level of detail.

In our tests, master and slaves exchanged arrays of integers. We used 32-bit integers and we are keenly aware that this is a good fit for the 32-bit architecture of our processors. To gain insight in the impact of this, we re-implemented and ran our test using 64-bit integers so that MPI_Send and MPI_Recv had to do extra work. Indeed we observed that whilst 8 processes on 8 slaves took 32 seconds to search 1 .. 504000, the same task now took 53 seconds when 64-bit integers were used.

4.3. msgCourier

msgCourier is currently designed to leverage compute farms which are not necessarily configured as a Beowulf cluster. In this scenario, msgCourier is responsible for message passing in place of MPI.

We're currently exploring the potential of modifying msgCourier to leverage MPI when it is available on the target cluster.

5. Conclusion

This paper examines our early exploration into extending ChessBrain II to leverage the use of clusters. In particular we're beginning to explore Beowulf clusters and MPI for potential speed improvements.

Preliminary tests using prime number searches indicate that compute-intensive and easily-parallelizable algorithms stand to gain much from Beowulf clusters. ChessBrain II relies on such algorithms.

We propose the msgCourier technology as a partial solution to the problems faced by new volunteer computing initiatives, namely:

- Improving the effective utilization of clustered resources within local networks.
- Simplifying the creation of custom clustering software with minimal dependencies on the underlying operating system platform.

- Devising an underlying framework (msgCourier) on which to build distributed computing solutions, such as work distribution servers, gateways and distributed logging facilities.

We're preparing msgCourier to harness distributed clusters in support of volunteer computing and Grid computing initiatives.

6. References

[Gropp *et al* 1999] William Gropp, Ewing Lusk and Anthony Skjellum. Using MPI: Portable Parallel Programming with the Message-Passing Interface. MIT Press, 1999.

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

[Justiniano 2005] Carlos Justiniano. *Tapping the Matrix: Revisited*.
http://www.chessbrain.net/documentation.html

[Justiniano 2003] Carlos Justiniano. *ChessBrain: A Linux-Based Distributed Computing Experiment*. Linux Journal (13), Sept 2003.

[MPI 2005] MPICH – A Portable Implementation of MPI. http://www-unix.mcs.anl.gov/mpi/mpich/.
OpenSUSE 2005. The OpenSUSE Project. http://www.opensuse.org.

[OpenSuSE 2005] http://www.opensuse.org

[Sterling *et* al 1999] Thomas Sterling , John Salmon, Donald Becker, Daniel Savarese. How to Build a Beowulf: A Guide to the Implementation and Application of PC Clusters (Scientific and Engineering Computation), MIT Press, May 1999.

[Swendson 2004] The Beowulf HOWTO. http://www.tldp.org/HOWTO/Beowulf-HOWTO/.
2004-05-17.