The ChessBrain Project – Massively Distributed Inhomogeneous Speed-Critical Computation

C. M. Frayn
CERCIA, School of Computer Science, University of Birmingham, Edgbaston, Birmingham, B15 2TT, UK
C.M.Frayn@cs.bham.ac.uk

C. Justiniano
ChessBrain Project, Newbury Park, CA, USA

cjus@chessbrain.net

The ChessBrain project was created to investigate the feasibility of massively distributed, inhomogeneous, speed-critical computation on the Internet. The game of chess lends itself extremely well to such an experiment by virtue of the innately parallel nature of game tree analysis. We believe that ChessBrain is the first project of its kind to address and solve many of the challenges posed by stringent time limits in distributed calculations. These challenges include ensuring adequate security against organized attacks; dealing with non-simultaneity and network lag; result verification; sharing of common information; optimizing redundancy and intelligent work distribution.

1 INTRODUCTION

The ChessBrain project was founded in January 2002. Its principal aim was to investigate the use of distributed computation for a time-critical application. Chess was chosen because of its innately parallelisable nature, and also because of the authors' personal interest in the game. (Justiniano (2003), Justiniano & Frayn (2003))

Development of ChessBrain was privately financed by the authors, and was developed primarily over the last few months of 2002. It played its first complete game against an automated opponent at the end of December 2002. In the subsequent thirteen months, much work was done in refining the distribution algorithms and in improving the stability and ease-of-use of the freely downloadable client software.

In this paper, we introduce the algorithms used in computer chess to analyse a given board position, and we explain how these initial principles were expanded to a fully distributed framework. We also discuss the relevance of such work to the field of distributed computation in general.

2 COMPUTER CHESS

The first algorithm designed to play chess was informally proposed by Alan Turing in 1947, though no computers on which it could be tested yet existed. Undaunted, Turing learnt to evaluate the algorithm in his head for each move. This was probably the first ever example of a chess game played by formula. Unsurprisingly he didn't meet with much success, though within a few years Turing was already considering the possibility that a computer program might be able to out-play its creator (Turing, 1952).

The science of computer chess has therefore been in existence for a little over half a century. A computer first beat a human in 1958, though the opponent was an untrained secretary who had been introduced to the game just a few minutes earlier. It was not until 1977 when a Grandmaster first lost to a computer in a rapid game, and in 1992 when the then World Champion, Garry Kasparov, lost in speed chess to the programme Fritz 2. Most famously, in 1997, the IBM supercomputer Deep Blue beat GM Kasparov in a high-profile match at standard time controls. Just seven years ago, the fastest, most expensive chess computer ever built (Hsu (1999)) finally managed to beat the top human player.

The goals behind the ChessBrain project were very clear. Firstly, we wanted to illustrate the power of distributed computation applied to a well-known problem during an easily publicised challenge. Secondly, we wanted to investigate the extent to which we could extend the boundaries of distributed computation in the direction of speed-critical analysis. Along with these motives, we also wanted to make a system capable of holding its own against the best in the world and, ultimately, of outperforming them.

Turing's original algorithm for searching a chess position was very simple, and forms the basis to the algorithms used today. The basic principle is very easily understood: "Any move is only as good as your opponent's best reply." This is a unique facet of zero-sum games such as chess and draughts, and allows us to evaluate a game tree unambiguously.

To analyse a particular board position, we simply make a list of each potential legal move. We test each of these moves in turn, arriving at a resulting board position, from which we list the opponent's legal replies. At any one position, the node value is equal to **minus** the value of the opponent's best reply. That is to say, a move that is very good for white is necessarily very bad for black, and vice versa. This process is called a *minimax* search, meaning that we are maximising the score for one player, and therefore minimising it for the other.

The minimax search algorithm allows us to build up a game tree consisting of the legal moves, and replies to those moves, and so on as deep as we wish to search. We then terminate the algorithm either at a predetermined depth or, more usually, after a fixed length of time, and assign values to the leaf nodes by performing a static evaluation of the resultant board positions.

Of course, this is an exponential progression. The game tree will expand in size by a factor equal to the number of legal moves from each position. For an average middle-game position, this number is approximately 35. That means that the size of the game tree quickly becomes too large to search.

There are several algorithms used by the top chess programmes in order to overcome the problem of exponential game-tree growth. Most of them work by reducing the branching factor of the tree at each node. The most obvious, and vital, of these is **alpha-beta search** (Newell, Shaw & Simon (1958)). This encodes the rule that "any branch that cannot possibly affect the minimax value of the current node can safely be pruned entirely". Without exploring the details of this algorithm, in summary it reduces the branching factor at each node to approximately the square root of its previous value, and therefore doubles the depth to which we can search in the same, fixed time.

At the top level, therefore, chess analysis reduces to a question of how quickly one can search a large game tree, and how much of the game tree can safely be pruned without affecting the result. In practice, all programmes today use theoretically unsound heuristics to prune the game tree further than *alpha-beta search* allows, though doing so introduces the element of uncertainty and chance into the analysis - it becomes theoretically possible that the optimum move within the game tree might not be found. The negative possibilities are balanced against the positive gains in order to assess the potential value of such heuristics.

Example heuristics are Null-move search (Beal (1989)), futility pruning (Heinz (1998)) and razoring (Birmingham and Kent (1977)). There are also some other algorithms that, whilst theoretically sound, are not guaranteed always to reduce the time required to search a given position and may cause a worsening of search speed in some cases e.g. principal variation search (Reinefeld (1983)) and MTD(f) search (Plaat (1996)).

3 DISTRIBUTED CHESS ALGORITHMS

3.1 Internal Structure

By thinking briefly about how a human analyses chess positions, it becomes obvious why this game is so well suited to distributed computation. The outcome of the analysis of one move from a given position is totally independent of the outcome of the analysis of its sibling nodes. The only caveat to this statement is that lack of knowledge of sibling nodes severely cripples our ability to prune the upper parts of the game tree. It is expected that the extra number of CPUs applied to the task will more than compensate for this. Indeed, distributed chess computation becomes a problem of minimising the drawbacks as much as possible and compensating for the unavoidable ones by strength in numbers.

In order to implement a distributed chess search algorithm, we created the following framework (Fig. 1). The central *SuperNode* coordinates the efforts of many *PeerNode* machines connected through the Internet. Each component comprises many separate parts, which will be covered in more detail in section 4.

The chess board analysis is performed in the central server by the *BeoServer* application, and on the PeerNode machines using the *BeoClient* engine. Both these are based on the existing open-source chess engine written by CMF and contain identical analysis and search algorithms. All PeerNodes are given exactly the same analysis and evaluation parameters, which ensures that the results from PeerNodes searching the same position agree with each other. It also allows the SuperNode to predict the time complexity of each work unit.

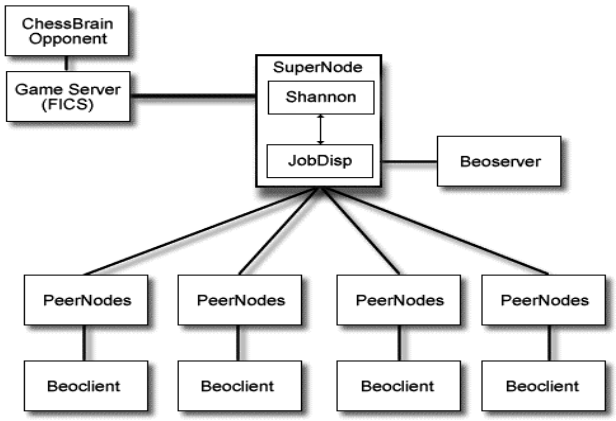

Fig 1: Simplified schematic of the ChessBrain architecture.

3.2 Positional Search

The ChessBrain project borrows its core distributed search algorithms from the APHID algorithm (Brockington, (1997)). It implements an incremental, iterative deepening search, firstly locally on the server and then, after a certain fixed time, within the distribution loop. During this latter phase, the top few ply of the search tree are analysed repeatedly, with new leaf nodes being distributed for analysis as soon as they arise. Any information received from the PeerNodes is then incorporated into the search tree, with branches immediately being extended or pruned as necessary.

Leaf nodes are distributed to PeerNodes as work units. These encode the current position to be analysed and the depth to which it should be searched. Work units are distributed to the connected PeerNodes on a request basis, though they are also ranked in order of estimated complexity using intelligent extrapolation from their recorded complexity at previous, shallower depths. In this way, the most complex work units can be distributed to the most powerful PeerNodes and vice versa. Work units that are estimated to be far too complex to be searched within a reasonable time are further subdivided by one ply, and the resulting, shallower child nodes are distributed instead. Work units which become unnecessary are deleted.

4 CHALLENGES UNIQUE TO SPEED-CRITICAL DISTRIBUTED COMPUTATION

4.1 Local Search

The overwhelming majority of work in the literature has considered distributed computation as an unhurried problem. That is to say, we are not concerned *when* we obtain results from various computations, or in which order they arrive. Provided, that is, that we complete the calculation within an acceptable time span, say a few hours or days. For chess, we have a very tight time limit imposed on every move, and this introduces a large set of challenges that must be overcome.

The most obvious challenge posed by speed-critical distributed computation is simply that we must ensure that we achieve a sensible result within the designated time limit. Our first task is to do just this – to obtain a result that is at least satisfactory so that, in our worst-case scenario, we don't play a terrible move.

To do this, we begin the search locally, analysing for a fixed length of time without distributing any work units whatsoever. During this period, it may be that an easy winning move is discovered; alternatively, we may be running out of time, meaning

that a distributed search is not wise due to the inevitable time overheads involved.

However, if we have sufficient time left, and the local search does not reveal any straightforward winning moves, then the distributed search begins, but we already have a result from a limited local search that could be used if the distributed search fails to locate and verify a definite improvement.

4.2 Job Duplication

Once the distributed search has commenced, it is necessary to ensure that we optimise our chances of recovering results from all the vital work units as soon as possible. We use three separate techniques to ensure this.

1. We distribute each work unit to more than one PeerNode machine.
2. We distribute nodes that appear to be the most important with the highest priority.
3. We estimate the complexity of each work unit before it is sent to PeerNodes so that we can send those that require the most computation to the fastest PeerNodes, and *vice versa*.

This is where the intrinsic difference between internet-based distributed computation projects and many traditional distributed computation projects is most clearly highlighted. The PeerNodes are of greatly varying internal specification and connection speed. Some PeerNodes will be brand new, high performance research machines, and some are ancient UNIX servers. Not only does the software have to work perfectly across these diverse platforms and architectures, but it should also optimally utilise the available resources.

In some circumstances, we are also required to deal with the possibility that a particular PeerNode might have been switched off, crashed or disconnected while processing a work unit. In these cases we must ensure that the work unit is duplicated on other PeerNodes if we wish to receive a result. Otherwise, the analysis could be delayed forever waiting for an extinct PeerNode.

When considering job duplication, there is a trade-off between ensuring that each work unit is sent out enough times to ensure it is returned efficiently and quickly, whereas at the same time, avoiding excess duplication - we have a lot of work units to evaluate and only a finite amount of time in which to do so. In practice, each node is sent out to a fixed number of initial PeerNodes, though at a later time, more PeerNodes can be assigned to the task if a result has not been achieved within a reasonable time frame. These values are all open to optimisation.

4.3 Security Issues

Chess is a highly unstable analysis problem: One single erroneous result could possibly corrupt an entire search and render any conclusion unreliable. It was therefore critical in ChessBrain that we dealt with security issues very carefully.

The first major consideration that we implemented was to remove all precalculated data files from the PeerNode software package. BeoClient was compiled with a hard-coded parameter set that represented a safe, well tested selection of options and algorithms that we were confident could produce reliable results. End game and opening searches were performed on the server, where only the project development team had access.

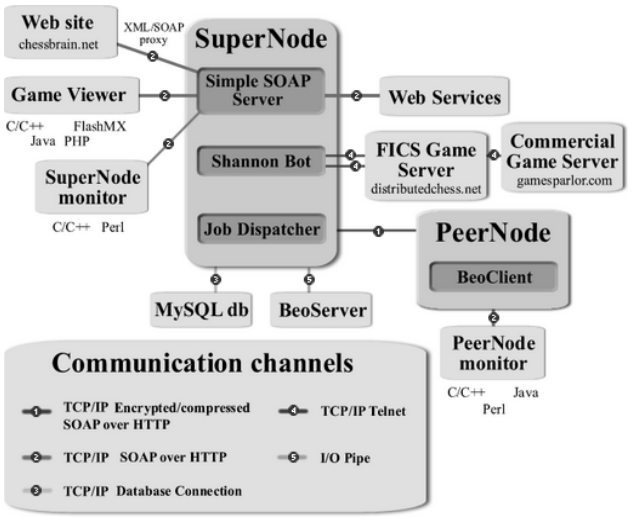

Fig. 2: Detailed overview of the ChessBrain architecture, outlining the many communication channels.

Another obvious attack point was the linkage between the chess analysis engine and the communications software in the PeerNode client. It was decided early on to join these two elements together in one executable file to avoid the extra communications overhead, and to remove another exposed pipe that could easily be exploited.

Figure 2 shows the many communication channels that exist within the ChessBrain architecture, several of which presented significant security issues. Communications between the PeerNodes and the central server were sent using encrypted SOAP messages over HTTP using TCP/IP. The encryption used is the industrial strength Advanced Encryption Standard, formally known as *Rigndael*. Each PeerNode client has a different encryption key, which is built using characteristics of the client system. Cracking the encryption key on one PeerNode will not compromise the hundreds of remaining nodes.

The SuperNode and PeerNodes also compress data using the *Zlib* compression library (written by Jean-loup Gailly & Mark Adler). Data is first compressed and later encrypted, giving a second layer of protection against malicious attacks.

In addition, the client software was supplied with MD5 checksum values (Rivest, 1992) and GnuPG (Gnu Privacy Guard) digital signatures, so that PeerNode operators could verify the unaltered condition of any software supplied by the ChessBrain team.

5 PRELIMINARY RESULTS

5.1 ChessBrain vs. GM Peter Heine-Nielsen

On 30th January 2004, ChessBrain played its first publicized game against a rated human player under an official judicator. This was the first time a fully distributed chess project had ever completed a game in such conditions. Our opponent was top Danish Grandmaster, Peter Heine Nielsen. A total of 2,070 individual machines from 56 different countries around the world participated in the event, which resulted in a draw after 34 moves. Individual machines were identified using the unique combination of IP address and Ethernet MAC address

5.2 Dealing with network overloading

As chess is a speed-critical application, with lots of work units being distributed and returned within a few seconds, the network

load is high. ChessBrain was initially tested in a LAN setup where this did not pose a problem. Indeed, until 24 hours before the Copenhagen event, the record number of distinct machines connected during any single fifteen minute period was just 846, and this was itself a recent result.

During the world record attempt, we logged a total of 2,070 separate contributors. The maximum number of concurrent contributors was slightly less than this number because some PeerNode operators did not remain connected for the entire match. As we had never tested the infrastructure at anywhere near this level of network activity, we unsurprisingly encountered some very serious challenges. Indeed, this first major test for the ChessBrain project proved nearly fatal for the entire event. In effect, we were suffering a severe distributed denial of service attack, but with the extra caveat that we had actually requested it!

In order to deal with this problem, we experimented with a much more severe PeerNode control algorithm. When there is insufficient work for the connecting PeerNodes they are forced to disconnect for a fixed delay time, related to their CPU speed. The system we used in the game was far harsher than anything we had tested before, but the overwhelming volume of network traffic required this.

5.3 Game Statistics

The total number of useful nodes processed was 84,771,654,525; that is the number of nodes calculated for work units that were accepted by the central server. Many more nodes were processed in work units that were never delivered due to server connection issues, as well as those nodes that were aborted before a result was returned. ChessBrain used a total of 2 hours, 15 minutes of thinking time. Using just the useful returned data, this gives an average of 10.5 million nodes per second. By contrast, Beowulf analyses approximately 100,000 nodes per second in an average position on a single P4/2.8GHz machine. This means, in terms of raw node processing, that ChessBrain performed at the level of a single 280 GHz CPU.

These figures also tell us that ChessBrain's efficiency in terms of converting connected machines to raw processing power was approximately 100 / 2,070 = 5%. Improving the central server architecture so that the connection problems are resolved would improve this figure dramatically. We are working on a server redesign to overcome this challenge before we enter any more high-profile matches.

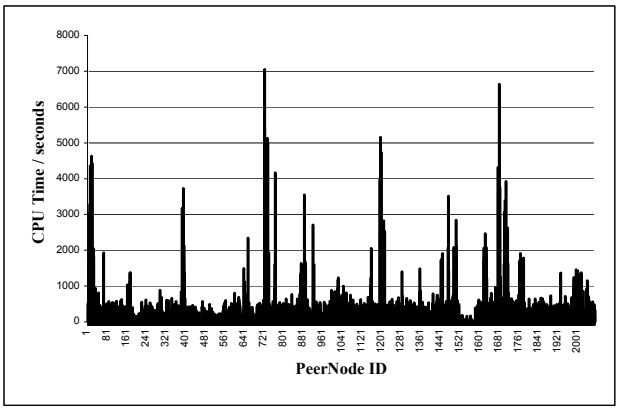

Figure 3: CPU time contribution for all PeerNodes.

Figure 3 shows the total CPU time contributed per user. Only 7.5% of users contributed more than 1,000 CPU seconds to the project, and only 13.3% contributed more than ten minutes (600 seconds). There was little correlation between contributed CPU time and processor speed – the principal variable here seems to be whether individual PeerNodes could connect to the server.

ACKNOWLEDGMENTS

CMF acknowledges the help of Advantage West Midlands in supporting his current research post. CMF and CJ are grateful to Kenneth Geisshirt and the DKUUG for bringing our record attempt to Denmark; ChessBrain member, Peter Wilson for his unwavering faith, support, and dedication; The Distributed Computing Foundation for organizational support and Y3K Secure Enterprise Software Inc. for their generous financial support.

Extra thanks go to Sven Herrmann and Farooque Khan for their work on the SuperNode monitoring software. This software uses *SQLite*, written by D. Richard Hipp et al. We acknowledge also use of the *AES-Rijndael* encryption standard and *Zlib* compression and encryption software.

And lastly, thanks to all the other members of the ChessBrain community who helped during the development period. They are too numerous to be listed here, but can be found at the following URL; http://www.chessbrain.net/wra/team.html

REFERENCES

Beal, D.F. 1989, *Experiments with the null move. Advances in Computer Chess 5*, (Ed. D.F. Beal), pp. 65–79. Elsevier Science Publishers

Birmingham, J.A. & Kent, P. 1977 *Tree-searching and tree-pruning techniques. Advances in Computer Chess 1*, (Ed. M.R.B. Clarke), pp. 89–107. Edinburgh University Press

Brockington, M. 1997 *PhD Thesis*, University of Alberta, Dept. of Computer Science

Heinz, E. A. 1998 *ICCA Journal*, Vol. 21, No. 2

Heinz, E.A. 1999 *ICCA Journal*, Vol. 22, No. 3, pp. 123–132

Hsu, F.-h. 1999 *IEEE Micro*, Vol. 19, No. 2, pp. 70–80.

Justiniano, C. & Frayn, C.M. 2003 *ICGA Journal*, Vol. 26, No. 2, 132-138

Justiniano, C. 2003 *Linux Journal*, September 2003

Justiniano, C. 2004, *O'Reilly Network Technical Articles*, 16th, 23rd April 2004. http://www.oreillynet.com/pub/au/1812

Newell, A., *et al*. 1958 *IBM Journal of Research and Development*, Vol. 2, No. 4, pp. 320-335

Plaat, A. 1996, *PhD Thesis*, Erasmus University, Rotterdam.

Reinefeld, A. 1983 *ICCA Journal*, 6(4): 4-14

Reinefeld, A. & Marsland, T.A. 1994 *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(7): 701-710

Rivest, R. L. 1992, *Internet Request for Comments*, RFC1321. http://userpages.umbc.edu/~mabzug1/cs/md5/md5.html

Turing, A. 1952, *The British Journal for the Philosophy of Science*, Vol. 3, No. 9.