CompTorrent: Applying BitTorrent Techniques to Distributed Computing

Bradley Goldsmith
School of Computing, University of Tasmania
bcg@utas.edu.au

Abstract

This paper describes "CompTorrent", a general purpose distributed computing platform that uses techniques derived from the popular BitTorrent file sharing protocol. The result is a grid swarm that requires only the creation and seed hosting of a comptorrent file, which contains the algorithm code and data set metrics, to facilitate the computing exercise. Peers need only obtain this comptorrent file and then join the swarm using the CompTorrent application. This paper describes the protocol, discusses its operation and provides directions for current and future research.

Keywords

Parallel computing, distributed computing, peer-to-peer computing, P2P, BitTorrent.

1. Introduction

Peer-to-Peer computing (P2P) and distributed computing are two separate fields in computer science that have many of the same goals variably based around the maintenance and control of a distributed pool of computing resources. P2P refers to a class of system which relies on equal (symmetric) peers rather than clients and servers (asymmetric) to consume and provide services respectively. Each node contains aspects of both the traditional client and the server. Classical distributed computing refers to a number of autonomous systems interconnected to each other to share resources [1].

Modern P2P has grown out of a grass-roots movement of individuals largely wanting to share files and information through incentives. Distributed computing for computation has grown out of the opportunity to harness multiple computers to provide a lower cost alternative to traditional super-computing hardware. Whilst these two fields have emerged for different reasons, both share many of the same problems of security, service guarantee, network maintenance/overhead and availability. Where they still differ is in ubiquity of the number of jobs being processed and some measure of symmetry between the number of instigators of jobs and the number of participants providing processor cycles. In file sharing networks, it is common for there to be many people both uploading and downloading content. That is, lots of people initiate a file sharing episode by uploading content which other people then consume. This contrasts to distributed computing where there might be many participants providing resources, there are not so many projects or jobs actually being instigated.

As an example to illustrate this we will consider two of the largest distributed computing projects on the Internet. One is the Berkeley Open Infrastructure for Network Computing (BOINC) [2] and the other Distributed.net [3]. Both of these projects allow people to participate by donating computing resources to one of a few projects available by these groups. Distributed.net hosts projects to challenge cryptography systems whilst BOINC is more diverse covering projects such as searching for evidence of extra terrestrials to medical computing projects. In each case, users download software and then join one or more of the projects that they want to contribute to. As at March 2006, Distributed.net listed 7 past projects with 2 current [4] and BOINC has 9 projects considered active [5] with a further 2 projects in beta testing [6]. So, whilst P2P file sharing and distributed computing share so many similarities in the problems that they face, they differ in the fact that so many jobs are instigated in P2P (files to be shared) yet so few are instigated in distributed computing (computing jobs).

There are potentially many more applications of raw computing power these days which may be of interest to a wider community. Not everyone wishes to start a project to factor prime numbers, but many may be interested in processing their home movies from raw video into a more compact format such as MPEG-4. Such algorithms are often very processor intensive where the time taken to distribute the original data is significantly less than the time taken for one average machine to process it. Organisations have expressed a desire for so-called Grid computing where a federation of computing resources is made available in the same way from multiple locations just like the power grid [7]. These kinds of applications involve an interested community of users pooling computing resources to share and process data that may number in the 10s, 100s or 1000s but perhaps not the tens of thousands or millions required to justify a large scale project such as SETI@home [8].

The individual requirements of a more ad hoc or smaller distributed computing job compares favourably to the BitTorrent scheme where separate user swarms exist for each individual job or file being shared. There is not one large community all sharing many files but a separate community (the swarm), is built around each file or set of files that is to be shared. This has resulted in a very distributed and efficient system where many of the overheads associated with maintaining a large, decentralised P2P network are avoided [9].

This paper introduces a new system called "CompTorrent" which aims to apply these desirable traits of the BitTorrent protocol to distributed computing. It is hoped that the same techniques that has made BitTorrent so ubiquitous can be applied to distributed computing to lower the "entry cost" of instigating a distributed computing job.

We begin by examining the features of the BitTorrent protocol in specific detail and then show how the CompTorrent protocol relates to it. In section two a description of the file format and protocol will be given followed by some results of preliminary application and experimentation in section three. To conclude, a summary of related work in the literature will be discussed as will the future direction of this work.

1.1 BitTorrent

BitTorrent is a novel approach to file sharing that aims to maximise the distribution of data in terms of bandwidth and speed [10]. When downloading a file with traditional FTP, a user only uses the download channel of their Internet connection whilst the upload channel is mostly unused. BitTorrent takes advantage of this and allows other users to download parts of the same file that another user is downloading at the same time. While a user is downloading a file they may also be uploading a file to another user using this otherwise unused upload channel. This balances the load of the file distribution bandwidth across all of the peers and reduces the load on the originating server. The original node need only upload one copy of the file being shared rather than FTP where the server needs to upload every copy requested by a client.

To use BitTorrent for downloading a user must first find a file which gives the necessary information about the swarm the user intends to join. As BitTorrent does not provide its own search facility it relies on existing search methods available to Internet users to find a "metainfo" file which contains the information required to join a BitTorrent session. This metainfo file, or "torrent", is typically hosted on an ordinary web server and is recognised by it's `.torrent` extension. The torrent file contains information about the file that is being downloaded, such as its length, name, number of pieces the file has been split into for distribution, hashes of these pieces, and the URL of a "tracker" to coordinate the swarm. Communities of "torrent sites" have appeared (and disappeared!) to provide a search service to users by hosting and categorising torrent files. Torrents are also commonly traded on other services such as Internet relay chat or the Usenet.

A BitTorrent tracker is a small HTTP based service that allows BitTorrent users using the same torrent to find each other. When each BitTorrent download is started, contact information about the new downloader such as their IP address, BitTorrent listening port, etc. are sent to the tracker. The tracker then responds with the same kind of information about others who are downloading the file to coordinate the swarm. New implementations of the BitTorrent protocol are also beginning to use distributed hash tables (DHT) in order to attempt a decentralisation of the tracker component [11].

BitTorrent also employs many techniques to ensure that the transfer between two nodes is as efficient as possible and that data is replicated as reliably as possible. Such techniques include pipelining of HTTP requests, order selection and prioritising of download chunks, choking algorithms and anti-snubbing of peers. Further information about the BitTorrent protocol, beyond what is needed to understand this paper, can be found at the wikipedia entry for BitTorrent [12].

1.2 Introducing “CompTorrent”

CompTorrent allows a user to distribute an algorithm and source dataset to any number of peers who wish to participate in a distributed computing exercise. A user wishing to compute something using CompTorrent need only create a comptorrent file containing an algorithm and description of a dataset and have it hosted where nodes can access it. The comptorrent file itself can be posted to a website, passed via email or any other appropriate method to advertise the computing job.

Once one or more nodes have obtained the comptorrent file and connected to the seed, each node processes allocated parts of the dataset and returns computed data to the seed node. The seed will ensure that each data subset is computed more than once by different nodes and the result compared. This is to help ensure that the resulting dataset can be trusted to the satisfaction of the instigating party. Once the seed determines the computation is complete, nodes can share the computed dataset amongst each other without the seed being necessarily involved. This approach allows an efficient computation and distribution of the dataset in the same overall process. Subsequent joining nodes need only share the data if they join after the computing has finished.

Computed data is replicated throughout known nodes as a part of the computation and confirmation process. Should computed data become lost to the overall swarm through node failure or churn, the swarm can revert back to re-computing and confirming the missing data portions from the original data.

2. The CompTorrent File Structure

The comptorrent file contains information regarding the seed node, algorithm and original data set. The file is formatted in XML to allow for easy extension and understanding.

```
<comptorrent>
<version>0.1</version>
<tracker_url>192.168.0.1</tracker_url>
<tracker_port>60000</tracker_port>
<name>randomdata</name>
<size>10482434</size>
<max_chunk_size>264000</max_chunk_size>
<md5>BCDFF226FC2430A0F36304CD66423122</md5>
<algorithm>
<execution>MyAlgorithm</execution>
<java_bytecode>
base64 encoded algorithm
</java_bytecode>
<classname>MyAlgorithm.class</classname>
</algorithm>
<orig_data>
<file><name>random_00000001</name><size>262
148</size><md5>369DE8B5FF6B4D32BEED580339F5D53
F</md5></file>
...
<file><name>random_00000040</name><size>258
665</size><md5>A2387C6E3D1ECEA1DB028333D0F5D36
2</md5></file>
</orig_data>
</comptorrent>
```

Listing 1. An abbreviated example comptorrent file.

The header section of the comptorrent XML contains the version of the protocol, the tracker url and port, the name of the computation exercise, the overall size of the data set, the maximum size each chunk of the dataset will be split into and a hash digest of the overall data set for verification.

The algorithm subset of the file contains the algorithm to be used for the computation (base64 encoded for transport within the XML), the class name of the algorithm and information on how to execute the algorithm (usually just the name of the algorithm class/executable along with any parameters). For now, the algorithm is assumed to be a platform independent java class file. Pre-compiled binaries for multiple platforms, or indeed source and makefiles, could be included just as easily from the point of view of the file format itself.

The original data subset contains information regarding each chunk of data in the overall set. The number of chunks that the data is split into is determined by the nature of the computation job. Each data chunk is represented by its name (the original data set name with an order number appended), the size of the chunk in bytes and a hash digest of the chunk itself.

A comptorrent file may be assembled by hand or produced with helper applications that have been written to split the files into data chunks and automatically produce the comptorrent file.

3. Protocol & Reference Implementation

The CompTorrent algorithm is simple and straightforward. A seed node hosts the original dataset and waits for a node to connect and participate in the computation. As nodes connect, the seed hands out computing jobs and tracks which data chunks have been allocated to which peer. When a peer has finished computing, by executing the embedded algorithm and applying it to the given original data, it sends back the computed data which the seed then marks as provisionally computed. The seed iterates through its list of original data allocating new work and receiving computed responses. During this process the seed will also allocate provisionally computed data chunks to a random node that is not the original node. This node will also compute and reply with the computed data. If the second (or *n* computation round as deemed necessary) computed data set matches the first set then the data is considered confirmed and is marked as such. Data sent for verification is not labeled differently so peers do not know if it is an initial computation or a validation computation. When this process has been completed for each data chunk, the seed will let each node know that the computation phase is complete and nodes can then request the entire original or computed data set. When asked, the seed will either reply with the data itself or direct the peer to another node on the network that is likely to have the data they want by sharing their IP address and port. The node can then connect directly to other nodes and request the data who then behave like seeds to that node. The original seed will attempt to balance requests between nodes that have the required data. This balances the distribution bandwidth to lessen demand on the resources of the seed node.

3.1 Protocol Specifics

All communication in the CompTorrent protocol occurs in the XML format. As shown in listing two, each message contains a packet identifier *packet* and then tags and data as defined by the context of the packet.

```
   <comptorrent>
     <packet>packet name</packet>
     <packet specific tag 1>data</packet
specific tag 1>
     <packet specific tag 2>data</packet
specific tag 2>
     ...
   </comptorrent>
```

Listing 2. CompTorrent message schema.

Each message has at least one corresponding reply. Table 1 shows each packet type in terms of statement and reply showing typical data.

`connect` This is sent by a peer to a seed (or another peer) after a tcp socket connection has been made. Data tags: `comp_chunks` is a tag which contains a string representation of which computed chunks the connection peer has. `uid` contains a global unique identifier (GUID) to identify this connection and thread. `host_ip` and `host_port` is the IP address on which this peer can be connected.	**`welcome`** This is sent by a seed (or another peer) after a tcp socket connection has been made and a `connect` accepted. Data tags: `uid` contains a global unique identifier (GUID) to identify this connection and thread.
`process` This is sent by a peer to a seed to indicate that it is prepared to accept a computing job if one is available. Data tags: None.	`process` can be replied to with either one of these messages: **`orig_chunk`** This is a message containing an original data chunk to be computed. Data tags: `chunk_name` is the chunk's identifier (corresponding to its filename). `data` is the actual data to be computed as defined by the algorithm. This would typically be CSV data or base64 encoded binary data. **`comp_finished`** This tells the peer that the computing phase is over. Data tags: None.
`request_comp_chunk` This is a message requesting a computed data chunk from a peer that has indicated that it has it. Data tags: `chunk_name` is the chunk's identifier (corresponding to its filename).	**`comp_chunk`** This is a message containing a computed data chunk. Data tags: `chunk_name` is the chunk's identifier (corresponding to its filename). `data` is the computed data as defined by the algorithm. This would typically be CSV data or base64 encoded binary data.

request_orig_chunk	**orig_chunk**
Data tags: `chunk_name` is the chunk's identifier (corresponding to its filename).	This is a message containing an original data chunk. Data tags: `chunk_name` is the chunk's identifier (corresponding to its filename). `data` is the original data chunk.
ping This is sent periodically by either a peer or a seed to test a connection and/or query what data a node has. Data tags: None.	**pong** This reply is sent back to indicate that the connection is live. Data tags: `comp_chunks` is a tag which contains a string representation of which computed chunks the connection peer has. `orig_chunks` is a tag which contains a string representation of which original data chunks the connection peer has.

Table 1: CompTorrent XML schema described in a statement/reply format.

These messages are traded between nodes in the network to facilitate the computing exercise.

3.2 Reference Implementation

The reference implementation of CompTorrent is a c++ application utilising the commoncpp, crypto++ and tinyxml libraries for socket communication, cryptography and xml functionality respectively. The code base is relatively portable and has been compiled on GNU/Linux and OSX with a Winodws port underway. Like the protocol, the application is simple and is built with ease of understanding and extension as its main focus. The application waits on connections and, if peering, will attempt to connect to the seed node indicated in the CompTorrent file. A separate thread is created for each of these connections. After a connection is made, each end of the connection takes turns sending and receiving XML data. Most of this functionality is provided by a class called `SimpleP2P` which provides a base from which a class called `CompTorrent` is derived which contains the CompTorrent protocol.

Each CompTorrent node can operate in either seed or peer modes. In both cases, a CompTorrent file is required which describes the original dataset and algorithm.

Each CompTorrent file has its own corresponding directory (named the same as the `name` element in the CompTorrent file) which contains two further subdirectories named `comp_data` and `orig_data`. When peering, the algorithm contained in the comptorrent file is decoded and stored in this directory for later execution. When seeding, the orig_data directory contains all of the data files corresponding to those in the CompTorrent file. When peering, the orig_data directory will minimally contain the orig_data chunks that the seed node has asked a peer to compute. These computed chunks will then reside in the comp_data subdirectory as they are created. As per the protocol overview, once the computing phase has finished, the peer may wish to assemble the complete original and computed datasets. These data chunks will then be stored in the same subdirectories as used in the computing phase.

4. Evaluation

Preliminary evaluation of the system was undertaken at the University of Tasmania, School of Computing. 16 Pentium 3 800Mhz machines with 256MB of RAM were utilised running Ubuntu Linux 5.10. Each machine was connected to the same network segment through a dedicated switch. Each system also used Java Runtime Environment 1.5.0 for the algorithm component of the comptorrent. A 17th machine of the same specification was used to seed the comptorrent file but did not participate in computation process.

A dataset consisting of 10Mb of random numbers was employed as a test with a simple numeric algorithm to load each machine. A SAXPY [13] loop was chosen as the algorithm for the experiment to test the implementation. Two small vectors (X,Y) were employed for the loop such that vector X was multiplied by each scalar in the dataset and then the resultant vector was added to the other vector Y. The experiment was set to compute the data only. No replication or re-computation was involved. The resulting computed data set was 5 times larger than the original 10Mb data set due to scalar input and vector output.

The result of this experiment was that the system worked as expected and average timing figures are illustrated by table 2 below.

Number of Machines	*Average Time to Run (sec)*	*Optimal Run Time*	*Std Dev of Run Time*	*Average Speed up*
1	115.43	115	0.52	1
2	65.29	65	0.46	1.77
4	35.43	34	0.93	3.26
8	18	17	1.13	6.41
12	11.29	11	0.52	10.23
16	8.86	8	1.73	13.03

Table 2: Preliminary results showing speedup factors per the size of swarm. Each experiment was conducted 8 times.

5. Related Work

Wei et al [14] have used BitTorrent as a means of coordinating large data sets for their Desktop Grid System. This work used BitTorrent as the distribution mechanism for data transfer and not any of the other aspects of the BitTorrent scheme i.e. the tracker. They found that BitTorrent had a large latency when compared to FTP but was significantly better when dealing with large files and large numbers of nodes. The BOINC project has also considered the use of BitTorrent for dataset transfer as a future goal [2].

Atkinson and Malhotra have contrived another system that uses a low cost entry to a distributed computing exercise using Java Web Start [15]. This system is a traditional client/server approach with a low cost entry twist: a potential computing node need only click a hyperlink on a web page to contribute processing cycles. No installation or downloading of separate software is required.

6. Conclusion and Future Work

This paper has briefly described CompTorrent, a general purpose parallel computing platform that uses several techniques derived from BitTorrent; namely the torrent file to attract participating nodes and the data distribution scheme to alleviate bandwidth requirements on the seed node. This research forms a part of the author's PhD research and as such is still very much a work in progress. The following future directions have been identified as extensions to the protocol and further research in swarm computing networks:

Support for more types of algorithms – The current protocol only supports algorithms that are purely parallel in nature. This is an obvious limitation that needs to be overcome in order to make the protocol more applicable to a broader range of algorithms.

Compression and Encryption of XML – Raw XML is very good for understanding and easy extension but takes up a lot more space than binary structure based message communication as commonly used in other P2P protocols such as gnutella [16]. Several XML compression specific techniques exist (Cheney notably [17]) that can provide smaller bandwidth requirements than text based compression alone. A hybrid approach may be suitable here as data contained in a message may compress differently to the rest of the XML message itself.

Security Features – Digital signing of the torrent file is a logical next step. This will allow for each participating node to authenticate the torrent file against the seeding node. This is in development now. In addition, symmetric or asymmetric encryption schemes could also be considered for the protocol for confidentiality of messages however this is probably of limited benefit for public swarms. However, extending the protocol to include a mesh authentication scheme would allow for authentication and non-repudiation of all messages without complete reliance on a seed node. This would be particularly interesting in a swarm where seeding work is shared between nodes over time. If a mesh authentication scheme were implemented the potential for a scheme of work ratio and credits could also be established. Nodes would then be able to have a track record with the network allowing some decisions to be based on some level of trust. This has already been explored in BOINC and some P2P clients such as Limewire and Azureus.

Sharing seeding work – The protocol only allows for one seed at present which is the seed for the duration of the life of the CompTorrent. This introduces an obvious point of failure. The ability to handover seed work from one seed to another is avoid this limitation. Extensions to the protocol are in early testing now which allows seeding work to be “handed over” to another peer node which then assumes the role of seed. An interesting feature is that losses of computed data are accounted for as the swarm will revert back into the computing phase allowing for the swarm to “heal” if computed data is lost (as briefly discussed earlier in section 1.2). Replication of original data as soon as possible might be a scheme worth investigating for volatile networks or where the seed wishes to hand over to another peer as quickly as possible. Seeding work could also be shared amongst nodes via distributed shared memory.

Node conspiracy – The protocol is particularly susceptible to conspiracies between nodes. Further work, beyond simple replication, should be considered to minimise the ability for nodes to contribute to the authentication of incorrectly or maliciously computed data. Consensus algorithms for choosing which node should do the authentication computation is under consideration now. This should provide an interesting avenue to investigate in concert with sharing seeding work via distributed shared memory or a distributed hash table.

Node categorisation – Each node is treated as being the same in terms of bandwidth and computing power when jobs are allocated. Obviously this is not the case in reality. It might increase overall effectiveness if jobs are matched to suitable nodes so that faster nodes do not sit idle whilst slower nodes do critical work. The potential may also exist for the "sub contracting" of jobs to smaller nodes akin to the super node and leaf node system used in some Gnutella implementations to attempt to improve routing times in its overlay network. This could allow slower nodes to be more effective to the overall network through some hierarchical arrangement.

Further improvements would also include the distribution of large algorithms by BitTorrent itself (to alleviate their mandatory embedding in the comptorrent file) as well as investigating the possibility of having the seeding be done via another dedicated seeding swarm. The provision of general services through a CompTorrent swarm such as seeding for other torrents, data backup or search indexing could potentially complement other general Internet services such as the domain name system or existing Internet search/indexing schemes.

10. Acknowledgment

The author gratefully acknowledges the assistance of his PhD supervisor, Dr. Daniel Rolf, for his helpful comments and suggestions.

11. References

[1] C. Leopold, "Parallel and Distributed Computing A survey of models, paradigms and approaches", pp. 2, Wiley, USA, 2001.

[2] D. Anderson, "BOINC: A System for Public-Resource Computing and Storage", 5th IEEE/ACM International Workshop on Grid Computing, November 8, 2004, Pittsburgh, USA.

[3] Distributed.net, http://en.wikipedia.org/wiki/Distributed.net, Accessed April 11, 2006.

[4] http://www.distributed.net/projects.php. Accessed March 15, 2006.

[5] http://boinc-wiki.ath.cx/index.php?title=Catalog_of_ BOINC_Powered_Projects. Accessed March 15, 2006.

[6] http://boinc-wiki.ath.cx/index.php?title=The_Current_ List_of_Alpha_Test/Beta_Test_Projects. Accessed March 15, 2006.

[7] I. Foster, "The Grid: A New Infrastructure for 21st Century Science.". Physics Today, 55 (2). 42-47. 2002.

[8] Wikipedia, SETI@home, http://en.wikipedia.org/wiki/Seti@home, Accessed April 11, 2006.

[9] J. Ritter, "Why Gnutella Can't Scale. No, Really", http://www.darkridge.com/~jpr5/doc/gnutella.html, 2001, Accessed April 11, 2006.

[10] B. Cohen, "Incentives build robustness in BitTorrent", in Workshop on Economics of Peer-to-Peer Systems, Berkeley, CA, USA, June 2003.

[11] Trackerless BitTorrent, http://www.bittorrent.com/trackerless.html Accessed March 15, 2006.

[12] Wikipedia, "BitTorrent", http://en.wikipedia.org/wiki/Bittorrent, Accessed April 11, 2006.

[13] J. Hennessy, D. Patterson, "Computer Architecture: A Quantitive Approach", pp 45-53, Morgan Kaufman, USA, 1990.

[14] B. Wei, G. Fedak, and F. Cappello, "Collaborative Data Distribution with BitTorrent for Computational Desktop Grids", *To appear in proceedings of ISPDC'05*, IEEE, Lille, France, 2005.

[15] A. Atkinson, V. Malhotra, "Coalescing Idle Workstations as a Multiprocessor System Using Javaspaces and Java Web Start". *Proceedings Eighth IASTED Intl. Conference on Internet and Multimedia Systems and Applications*, pages 233-238, Kauai, Hawaii, USA, 2004.

[16] "The Gnutella Protocol Specification v0.4", http://www9.limewire.com/developer/gnutella_protocol_0.4.pdf, Accessed April 11, 2006.

[17] J. Cheney. "Compressing XML with Multiplexed Hierarchical PPM Models", http://xmlppm.sourceforge.net/paper/paper.html, Accessed April 11, 2006.