Enabling Grassroots Distributed Computing with CompTorrent

Bradley Goldsmith

School of Computing, Faculty of Science, Engineering and Technology,
University of Tasmania, Australia
bcg@utas.edu.au

Abstract. This paper describes the operational characteristics of "CompTorrent", a general purpose distributed computing platform that provides a low entry cost to creating new distributed computing projects. An algorithm is embedded into a metadata file along with data set details which are then published on the Internet. Potential nodes discover and download metadata files for projects they wish to participate in, extract the algorithm and data set descriptors, and join other participants in maintaining a swarm. This swarm then cooperatively shares the raw data set in pieces between nodes and applies the algorithm to produce a computed data set. This computed data set is also shared and distributed amongst participating nodes. CompTorrent allows a simple, "home-brewed" solution for small or individual distributed computing projects. Testing and experimentation have shown CompTorrent to be an effective system that provides similar benefits for distributed computing to those BitTorrent provides for large file distribution.

1 Introduction

Distributed computing has had several high profile successes. Folding@Home [1], since 2000, has been working on computing simulations for molecular dynamics simulations to better understand certain diseases. Distributed.net [2] has spent the last 10 years answering challenges by RSA Security to encourage research into computational number theory. Also, arguably the highest profile project, SETI@Home has had over 5.2 million participants processing data from the Arecibo radio telescope making it the largest distributed computing project to date [3]. These projects, and others like them, are interesting, worthwhile and largely centralized in their control. This centralization has led to many participants, but relatively few distributed computing projects when compared to other distributed applications such as file distribution where more enabling software exists. There are potentially many more applications of raw computing power these days that could be pioneered with greater access to a pool of willing participants with available processors.

In light of this and inspired by the success of the BitTorrent file distribution system [4], we have set out to apply some of the techniques that have made BitTorrent successful to distributed computing. We have called this application

Samuel R.H. Joseph et al. (Eds.): AP2PC 2007, LNAI 5319, pp. 85–96, 2010.

CompTorrent and it forms a substantial part of the author's PhD research. In this paper we show that CompTorrent is relatively generic and easy to use for both joining and creating a parallel computing project. We start with section two below which explains what CompTorrent does and its overall operation before section three explains each major part of the system in some more detail. In section four we show and discuss some preliminary experimentation with the system before discussing ongoing work and future directions in section five.

2 What CompTorrent Does

CompTorrent allows a small group or an individual to host their own distributed computing project. This is achieved without needing to know much about distributed computing and, in many cases, without writing any new code. CompTorrent allows a group of nodes to share a dataset that needs to be computed. They share the original dataset, the computation load and the resulting computed dataset. This allows an originating node to upload an original data set only once and still share the entire dataset amongst many nodes. CompTorrent shows a decentralized peer-to-peer network being successfully used for distributed computing.

CompTorrent introduces several new techniques to distributed computing in order to solve some existing problems. Most importantly, and unfortunately also hardest to quantify, we claim to lower the cost of entry to distributed computing from the perspective of those wanting to have something computed. Joining a computing project tends to be easy. Starting one however requires much more work. Many systems, such as those mentioned in the introduction, have a very simple means of joining the system mainly the installation of some software and then the running of an application; often presented as a screen saver for when the machine is otherwise unused. Others based on Java Web Start [5] for example, can be joined with the click of a URL. Any earlier difficulties perceived in joining a distributed computing project have very much been solved. However, the creation of a distributed computing project tends to be more difficult. The Berkeley Open Infrastructure for Network Computing [6] (BOINC), arguably one of the more open and easier systems to create with, still requires the dedication and configuration of server hardware to the task of managing a project. The Gnutella Processing Unit (GPU), another distributed computing project, this time more P2P in nature than the BOINC client/server model, does not natively allow your own projects to be created at all [7]. Many other systems exist that are dedicated to a particular task and would fall into the very difficult to create category i.e. if you want to start a distributed computing project then first you must write a distributed computing system. CompTorrent, whilst not the first to introduce a generic distributed platform, it is the first to utilize the tracker and "metadata file" concepts to attempt to satisfy the goal of making a system that is both easy to join and easy to create new projects.

To provide an overview of this new system we start with the notion of a "seeder", that is, the group or user who initiates the distributed processing task

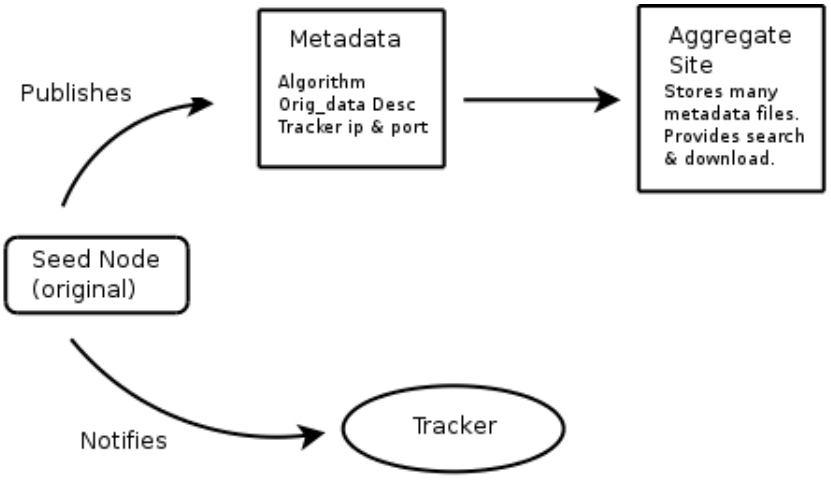

Fig. 1. A new job is started by the creation and publication of a metadata file

and has a full set of the original data, creates a metadata file (using a software tool) which describes or contains the algorithm and describes the data set. This metadata file can be published on the World Wide Web (WWW) or another peer-to-peer (P2P) service, for interested parties to download. This process is illustrated in figure one below.

The distributed metadata file is what allows other users to join in the computing exercise. Once downloaded, another interested user uses the CompTorrent application to read this metadata file, extract the algorithm, begin computation and attempt to join the other computers working on the project. It does this by first contacting a "tracker" whose contact details are included in the metadata file. The tracker is a service hosted on the WWW which maintains information about which nodes are currently working on a problem and which parts of the problem are currently unsolved. The tracker suggests tasks for each node and helps coordinate the process. Its serves as a shared memory for the swarm and does so independently of it. This process is illustrated in figure two. We claim that these techniques greatly simplify the task of starting a distributed computing project whilst also leaving it equally simple to join. It also allows separate computing jobs to be completely independent of one another so as to minimize any overhead in maintaining any other project other than your own. This approach contrasts with having a large group of nodes running multiple projects divided between them. Using file sharing as an example, BitTorrent's approach of a single swarm per file set easily out paces Gnutella's approach of one large network with many file sets [8,9].

CompTorrent also introduces the notion of sharing the data set as well as the computation at the same time. Whilst a distributed system has always needed to share some of the data, namely the data being computed, here the incentive to join a project can also be to share in the original as well as computed data.

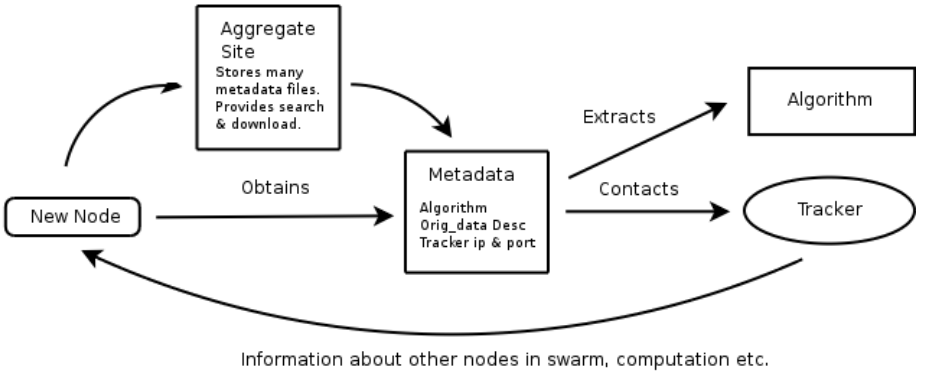

Fig. 2. The major steps in joining a CompTorrent swarm

Collaborative video encoding from a higher to lower bit rate can share the work and distribute the result at the same time. Using the output of one computing exercise among several research groups for the input of another is also a tangible incentive.

3 Technical Description

Following the overview given in the previous section we will now describe each major part of the CompTorrent system in more detail and show how each part interacts with the rest of the system. Security and trustworthiness of the overall system is also discussed.

3.1 Metadata File

The metadata file contains information about the location of the tracker, the algorithm to be used and a description of the original data set. It contains everything a new node needs to find the tracker to join the swarm, the algorithm used to compute a part of the result and the sizes, names and hashes of each piece of original data to be computed. This file is formatted in XML. An example is given in listing 1.

The first section of the file contains the version of the file, the connection details for the tracker, the name of the computation project, the size and hash of the original data set. The algorithm subset of the file contains the execution details of the algorithm and the algorithm itself. There are two broad options for the algorithm in the metadata file. The swarm can rely on the algorithm application being available on the participating machines (as is shown in the example given) or the application binary can be directly embedded into the metadata file as a base64 encoding. Either way, this approach allows the seeder to distribute the algorithm as flexibly as possible. Java bytecode is easily included or

a more complicated script can be used to broadly cater for a variety of situations and platforms. Once the algorithm has been extracted or obtained, the execution field stipulates how the algorithm is executed. It is assumed that there will always be an input file that contains data that will be acceptable to the algorithm. A resulting computed data set will be produced and saved in a computed directory.

Listing 1. An abbreviated metadata file

```
<?xml version="1.0"?>
<comptorrent>
<version>0.1</version>
<tracker_url>144.6.40.251</tracker_url>
<tracker_port>60000</tracker_port>
<name>cruelcruellove</name>
<size>93130756</size>
<md5>8EE44CB5C9A5AFCACD6C0AF363C1C5A1</md5>
<algorithm>
<execution>algo.sh</execution>
<script>
#!/bin/sh
transcode -i $1 -o $2 -y xvid
</script>
</algorithm>
<orig_data>
<file><name>chunk-001.mpg</name><size>1035738</size>
<md5>055172279073E1DC42C847BC794816A5</md5></file>
...
<file><name>chunk-100.mpg</name><size>703680</size>
<md5>BCEB81C97C89B6C0D61CFC8F8F1384DE</md5></file>
</orig_data>
</comptorrent>
```

The remaining section of the file describes the original data set. It does not contain the original data, only its representation in terms of name, size and hash. Nodes ask each other for original data as necessary and share the bandwidth load of the distribution task. The size of each data chunk in the set is dependent on the nature of the job and left to the judgment of the seeder at the time the metadata file is created. A typical data chunk size may be in the range of 256kb to 1Mb depending of the intensity of the computation task and quality of the network connection between typical nodes. Just like the algorithm, this data can be sent in plain text or Base64 encoded depending on the nature of the data to be processed. Anything that is acceptable to XML can be left in original form whilst binary data can be encoded or compressed and encoded. Along with each chunk of data, the size of the data and a hash of the data for checksumming is included.

3.2 Tracker

The tracker is a WWW service that provides a simple shared memory for a swarm or number of swarms. From the tracker a node can get a list of other connected nodes in the swarm, get a suggestion for the next data chunk to process and report data chunks finished. It is a simple service that is basically a web-based front end for an SQL database to allow nodes to gain and provide information quickly. A tracker is kept simple and provides no significant processing services so that a swarm need not completely rely on it for its work. As such a node does not necessarily keep an open connection to the tracker at all times. It connects and makes requests as needed. The bandwidth requirements for a tracker are also low and there is no reason why the tracker could not be ported to other mediums beyond HTTP. This is the subject of ongoing work and is discussed later.

As this is an ongoing research project and accurate results are required, the current implementation of the tracker includes tools for gathering and disseminating as much operational data as is possible. Data is made available on which nodes have which data pieces and at what time files were received or calculations made and at what times were connections between nodes made. All of this data is displayed in a web based application. Real time graphs of network topology are available as are visual indicators of original and computed data per node.

3.3 Node

Nodes speak a simple protocol that is represented in XML and communicate via sockets. This protocol is original and is not compatible with any other peer-to-peer protocol. XML was chosen due to ease of ongoing modification to the protocol when compared with a binary message structure style approach. Nodes make connections with each other after asking a tracker for nodes that are already in the swarm and how many existing connections to other nodes they already have. Each node may make many outgoing connections to other nodes and receive incoming connections as well. This overlay network is maintained for the life of the swarm as new nodes join and existing nodes leave. Presently, connection candidates are suggested by the tracker based on the simple heuristic of choosing the least connected node from a pool of nodes that do not have connections which involve the new potential node. A routing scheme is then overlaid over the underlying TCP/IP network.

The communication protocol is simple and largely consists of messages to manage connections and exchange data chunks. Connection requests include information about what original and computed data a prospective partner node has and details about which other nodes it is already connected to. A node may then accept or refuse a connection with a reply and pass back similar information to take advantage of this brief connection. Connected nodes pass file request and file reply messages back and forth as they work towards completing their datasets.

A node computes a part of the overall job and reports to the tracker that it has finished. Nodes make requests to each other to ask for parts of the original

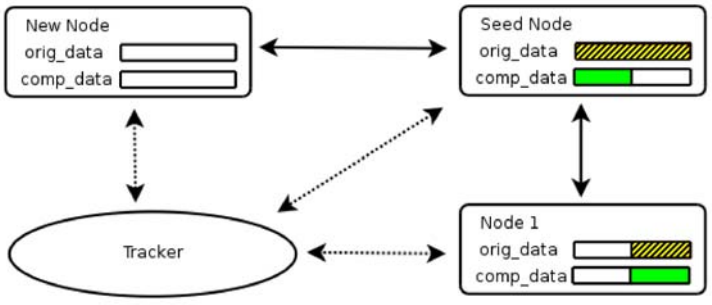

Fig. 3. Three nodes is a simple network interacting with each other and the tracker

and the computed data sets. Once a node obtains a new chunk of data, it reports this to the tracker so it can service requests for that chunk as well to help share the load.

Each node is equivalent to every other node in the network and has no different functionality whether it be an originating seed or a new node joining a large existing swarm. Every node computes and shares data with every other node. There are no "special" nodes with greater importance to the swarm or different responsibilities. It is the equal aim of each node to assemble, and maintain, a complete set of both the computed and original data sets in an attempt to provide as much redundancy as possible to the swarm as a whole. Computed chunks that are lost can be recalculated allowing the swarm to heal itself should critical nodes leave. Original data is replicated quickly amongst nodes in a rarest first fashion. To further illustrate this process, figure three shows a seed node with a full original data set and half of the computed set. Node 1 has obtained half of the original data and applied the algorithm producing some computed data. The tracker helps direct each node to form an overlay network and suggests chunks for computation and sources for data. As the computation is finished, the new node would work with the seed and node 1 for copies of the original and/or computed data.

The reference implementation is written in c++ and utilizes the commonc++, crypto++ and tinyxml libraries. The application and all of the libraries used are compilable with gcc ensuring that it is relatively portable between all major platforms. Both the application and tracker are completely original code and share no similarity with any other peer-to-peer code base.

3.4 Security

Whilst not actually a discrete part of the system like the tracker or the node, the security implications of the system need to be considered in order to gauge its usefulness. From the description of the system already given, it is clear that the behavior of nodes can have a dramatic impact on the reliability of the system. Whilst the subject of ongoing research and implementation, we can discuss the main features addressing these issues.

To begin with there is firstly an implicit trust in the seeder or the group who has constructed the metadata file and a new user wishing to participate. This is especially the case when a custom or unknown algorithm is the agent of computation. This is where the ability to include a script to use an existing application might be more suitable. It is envisaged that in time, users would be able to gain credibility, based on the quality and trustworthiness of their offerings, that would be manifested in a community that has grown around the distribution of the metadata files itself. This is certainly what has occurred with BitTorrent where there are many search or aggregation sites which serve as databases for existing BitTorrent swarms. Users are commonly allowed to make comments on each file available.

Once there is some measure of trust in the seeder, the integrity of the metadata file itself, whilst not currently implemented, could be managed with a digital signature scheme using existing tools. Original data integrity is already managed with hashes and this would obviously be further strengthened if the metadata file, containing the original set of hashes, was digitally signed by the author.

The computed datasets are clearly candidates for malfeasance. The hashes given in the metadata file protect the original data, but that does little to suggest the integrity of their computed equivalent. In CompTorrent, a seed may stipulate how many times each data chunk is to be recomputed, by a separate random node, before it is considered trustworthy. This clearly has a profound effect on the time needed to compute an entire set however in an uncontrolled environment it is one of the few tangible ways to get an idea of how much trust can be placed on a result. Other distributed systems commonly use various credit systems, such as ratings in online forums, to manage and rate node contributions, this is something that may be examined in the future for inclusion.

4 Evaluation

Preliminary results of an implemented system are available. Using 16 Pentium 3 machines (800Mhz, 256Mb RAM) running Linux (kernel 2.6.12). Each machine was connected to the same network segment.

The problem of recoding video was selected as an example of using an existing algorithm, in this case Transcode [10], where a reasonable amount of data could be processed using an algorithm that has a significant computing load. An arbitrary public domain movie (Charlie Chaplin's "Cruel Cruel Love") was selected as a candidate for conversion. This 93Mb file was broken down into pieces of under 1Mb each to result in 100 separate chunks to be processed. The data, given in figure 4, shows an expected linear result for the independently parallel problem.

Figure 5 shows the speedup based on the size of the swarm. The maximum run time on 1 machine was 18:31. It is interesting to note that another consideration is the time taken to initialize each algorithm run. Execution of Transcode natively on the original file on a single machine resulted in a run time of 16:10. Naturally, the overhead of the CompTorrent application is responsible for some of this

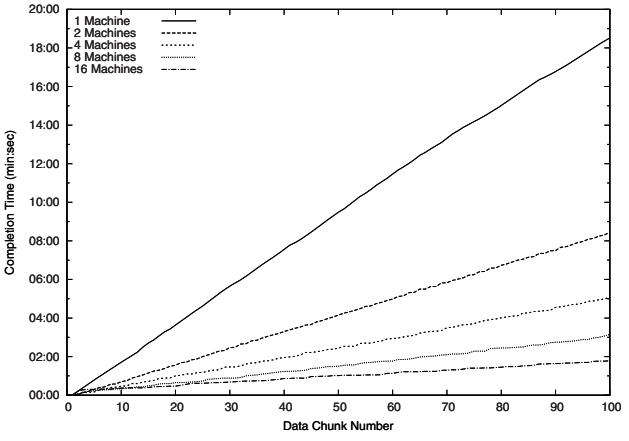

Fig. 4. Shows the time, in seconds from start, when each data chunk was computed

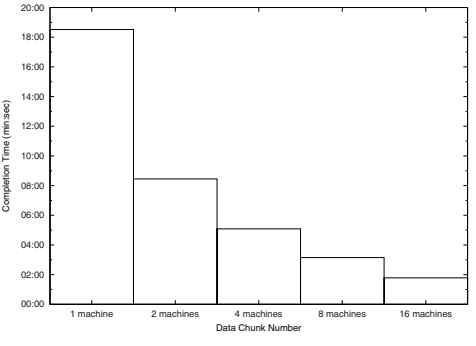

Fig. 5. Elapsed computation time for the computing job based on the number of participating machines

difference, however the time for Transcode to load and execute was observed to be approximately 1 second depending on the load of the machine. 100 loads on 100 chunks of data goes a long way to accounting for this 141 second difference and contributes to making a more precise measure of overhead more difficult.

4.1 Discussion of Network vs. Processing Time

Obviously the time taken to upload the full data set to the swarm versus the time taken to compute the result set will be paramount for the seed when considering the value of a CompTorrent exercise. Some more trivial applications of CompTorrent, such as recoding video, may take longer to upload and download the original and computed sets respectively. However, the overall time taken for the swarm as a whole to receive a computed set is determined by the upload speed of the original seed and the algorithm run time. So if the swarm as a whole

is interested only in the computed set, simultaneous computation and distribution may well get each node the computed set in less overall elapsed time when compared to single machine computation first and then distribution. Certainly the lag between the original data set becoming available and the start of the computed set being distributed is minimized with CompTorrent.

There are other incentives for a seed over just elapsed time as well. The load placed on the machine itself could also be a mitigating factor. The time taken for a single processor to complete a computation job obviously relies on the load placed on the machine itself. It may be reasonable to recode 4.6Gb of MPEG1 video to 700Mb of MPEG4 in 4 hours providing the machine is absolutely dedicated to the task. An 8 hour simultaneous upload, compute and download may be preferable where the machine is not highly loaded.

Also, as alluded to earlier, the load time of some algorithms must be taken into account. A dataset broken into small pieces will naturally incur more organizational overhead than a set with larger parts. Also the slow load or initialization time of some algorithms needs to be considered in overall time figures.

5 Ongoing Work and Summary

There are several areas where work is underway to further investigate P2P and distributed computing with CompTorrent.

The results presented in this paper are clearly what would be expected given the nature of the computation task. Further work is already well underway for comparing this system with other distributed computing platforms especially those where both server and client software is freely available. Examining the relationships between the nature of the computation task and the topology of the overlay network is already showing promise. Applying different routing algorithms is an area in its own right and further work beyond the least common ancestor heuristic which is used now should prove worthwhile. Other routing arrangements used in distributed hash tables such as a skiplist, Cartesian coordinate space, Plaxton tree, etc will be compared to see if they offer performance benefits as well as considering their cost in terms of implementation complexity.

Future work under consideration includes support for algorithms that are not completely independently parallel. The classic choices between shared memory or message passing are two obvious candidates for implementation and testing. Implementing shared memory across nodes in a CompTorrent swarm would also allow for a distributed tracker to be overlaid on the network. This could either be as a primary or secondary tracker service and it will be interesting to see how this could be used to improve the robustness of the system.

Optimization of file transfer is another area that will yield results. A lot of work has already been done investigating the efficiency of BitTorrent for file transfers including some recent work [11] that has further increased performance by some 70% by selectively uploading to connected peers based on their behavior. It will be interesting to see if these ranking algorithms would have a similar result with peers based on their bandwidth contribution as well as their computing

contribution. This would expand on the existing work of allocating work based on the number of data chunks processed, number of file requests services, time taken to respond, etc.

Tracker services beyond HTTP is another active area of investigation in this work. The tracker is currently a HTTP service and has a relatively small bandwidth load (subject to the granularity of the task and data). A recent idea involves investigating the possibility of embedding tracker data into unlikely places or protocols. As the tracker is mainly shared memory (lists of connected nodes, completed chunks) it may well be possible to host tracker data on another unrelated service such as Internet Relay Chat. An obfuscation technique that has already been proven in concept is embedding tracker data into an image using steganographic techniques. It will be interesting to see if the extra bandwidth required will result in any stealth advantage. As would looking at the mobility of projects between trackers during computation.

To summarize, we have presented a distributed computing system that is relatively generic and easy to use for both joining and creating a distributed computing project. We have shown that decentralized P2P distributed computing is possible for projects that can be computed with independent parallelism. We have applied techniques to distributed processing that have not been applied before, namely the metadata file and tracker paradigms, that have produced favorable results. This has allowed for CompTorrent to use many existing compiled programs without modification as an algorithm for a computing swarm to use. The operation of this system was demonstrated and sample results were provided.

Acknowledgments

The author would like to thank his supervisor, Dr. Daniel Rolf, for his valuable comments and suggestions. I would also like to extend my thanks to the anonymous reviewers whose comments improved the quality of this paper.

This research was supported by a Tasmanian Graduate Research Scholarship.

References

1. Larson, S.M., Snow, D.C., Shirts, M., Pande, V.S.: Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology. In: Grant, R. (ed.) Horizon Press Modern Methods in Computational Biology (2003)
2. Hayes, B.: Collective Wisdom. American Scientist 86(2), 118–122 (1998)
3. Anderson, D., Cobb, J., Korpela, E., Lebofsky, M., Werthimer, D.: SETI@home: An Experiment in Public-Resource Computing. Communications of the ACM 45(11), 56–61 (2002)
4. Cohen, B.: Incentives Build Robustness in BitTorrent. In: Workshop on Economics of Peer-to-peer Systems, Berkeley, CA, USA (June 2003)

5. Atkinson, A., Malhotra, V.: Coalescing idle Workstations as a Multiprocessor System using JavaSpace and Java WebStart. In: Proceedings of the 8th IASTED International Conference on Internet and Multimedia Systems and Applications, Kauai, Hawaii, USA, pp. 233–238 (2004)
6. Andreson, D.P.: BOINC: A System for Public-Resource Computing and Storage. In: 5th IEEE/ACM International Workshop on Grid Computing, Pittsburgh, USA (2004)
7. GPU: a Global Processing Unit Web page, `http://gpu.sourceforge.net/` (accessed January 25, 2007)
8. Bharambe, A., Herley, C., Padmanabhan, V.N.: Analyzing and Improving BitTorrent Performance. Tech. Rep. MSR-TR-2005-03, Microsoft Research (February 2005)
9. Ritter, J.: Why Gnutella Can't Scale. No Really, `http://www.darkridge.com/~jpr5/doc/gnutella.html` (accessed April 11, 2006)
10. Transcode Processing, `http://www.transcoding.org/` (accessed January 25, 2007)
11. Piatek, M., Isdal, T., Anderson, T., Krishnamurthy, A., Venkataramani, A.: Do incentives build robustness in BitTorrent? To appear in the 4th USENIX Symposium on Networked Systems Design and Implementation (2007)