Final Project Proposal

Ben Lipton Jonathan Gluck

March 24, 2010

1 Introduction

No system administrator can laud the usefullness of a reliable backup system enough. Backups add a measure of reliability to hard disks, which are an otherwise unreliable system. The Computer Science department at Swarthmore College maintains comprehensive snapshots of all network mounted directories; however, it does not maintain any backup of the local data on each of its lab computers. Disk space is cheap and plentiful, and in fact each lab machine has a great deal of disk space that is not currently being used. It seems a shame to do without backups when the the resources are readily available. The lab computers in question are connected in a broadband local area network, so bandwidth is another plentiful resource. We want a reliable backup system, so just buying a diskdrive and backing up all local directories to it would not suffice. If that disk failed, so would our backup. This is one major advantage of a peer-to-peer backup system. There is no centralized data, and if there is a level of redundancy then the chance of failure is reduced even farther.

Our lab is has peculiar requirements for a peer-to-peer backup system. Most groups of computers for which peer-to-peer backup has been implemented are scattered computers separated by the internet, owned by strangers, wanting to back up their own data and willing to donate as little disk space as possible to others to make that happen. They want to make sure their data will be available if they need it, even if there are poorly-behaved peers around that don't store all the data they are sent. Our situation is completely different. The machines are connected by a fast network, and we have control over all of them. That means that if a machine is working properly we can guarantee that it will be available when a restore needs to be performed; we won't have to wait for it to come online. Additionally, there are no malicious peers; all peers will be running the same cooperative software.

2 Related Work

Previous work on the subject of peer-to-peer backup has focused on squeezing security, reliability, and extra data storage out of untrusted peers. This requires encryption, spot checks to ensure that data is really being stored, and various systems for reducing the size of the backed-up data. Many systems, including pStore [1] and Pastiche [4] use convergent encryption, which ensures that identical data blocks will be encrypted identically, meaning they won't be stored extra times. Both of these systems make use of the Pastry distributed hash table. Pastiche uses it for routing, as a way to organize its peers, and pStore uses it for storing all the data. PeerStore [5] criticizes pStore for storing data in the DHT, and instead stores only metadata in a DHT, storing the actual backup data by trading data with peers via symmetric trading. The symmetry ensures that no machine backs up more data than it stores from other nodes, and also gives a node a way of punishing peers that don't seem to be keeping its data intact: it can delete their data too. This is one of the several methods these systems use to keep their peers honest.

One system that, like ours, doesn't use technological means to force peers to be available is Friendstore [7], which assumes that peers are linked by social relationships that would be strained by erasing backup data. As mentioned before, we are not worried that the software on our nodes will intentionally cause problems, but Friendstore also introduces an idea very important to our goals for the backup system. This is the concept of XOR coding, a tool they use to trade extra bandwidth for reduced disk space requirements and one we would like to implement in our system to increase the reduncancy of our backups without a huge increase in disk usage.

3 Solution

We hope to implement a robust periodic backup system making use of existing peer-to-peer infrastructure, including an established distributed hash table. This system will be used to back up the currently local-only **/local** directory on all of the department's public computers. We will design the system with the specific needs of our Computer Science lab's computers in mind, and consult with Jeff Knerr to make decisions that best suit the way the lab is used.

3.1 A Trusted Network

One of the factors wihch differentiates our network from those considered by other works is that all of the nodes are trusted. Many previous implementations have included complex policies to prevent tampering, reading others' data, and cheating by consuming more resources than one donates. None of these are issues in our case because the nodes all belong to this, and we will take advantage of that fact in our implementation. We are still considering whether or not we will want a form of encryption. This will be a consideration to bring to Jeff.

3.2 What will Define a Chunk?

The common terminology across most peer-to-peer backup implementations for the unit of backup is **The Chunk**. A chunk is a non-defined unit of data which will be backed up. How a chunk is defined varies. In our system we played with several ideas.

The first idea we toyed with was having one chunk per file. The problem with this path was that our distributed hash table would become laden down with too much metadata. It would be possible for the metadata to take up more space than the file it would represent, and the advantage of our system for using the hash table would be lost.

The second of our ideas was to take a literal chunk of non-content based data and index that as our chunk unit. For example, we could make a tar file of the directory, split it up into fixed-size pieces, and replicate them. The problem with this method is it reduces the utility of this backup to just rebuilding the whole local directory at once. There would be no way to only partially rebuild it to retreive one file.

Our third and current plan is to take as a chunk all files stored within a single directory. This will allow users to have some utility in only partial recoveries. If a user knew that he/she had stored a lost file in **/local/student1/img** then he/she could retrieve this directory's chunk from the system and subsequently retrieve his/her file. Though a little more complicated to implement robustly, this system gives a good balance between chunk size (presumably larger and therefore less wasteful for a directory than for an individual file) and usefulness (it allows retrieval of a specific file or the entire file system.

3.3 When will we back up?

We were left the question of how often do we want our system to perform backups. An initial idea touched upon the bandwidth reducing property of backing-up at incremental points based off of written data. *(e.g. five minutes after a 100kb write)* The issue with this implementation is that it required placing a layer between the user's programs and the system. We felt that this would carry serious overhead and was, if not beyond the scope of our abilities, at least too involved for the time we had available.

We decided upon an incremental backup at fixed time intervals during the day. This seemed to be the least invasive way for a system to backup the directories. When started up for the first time, the system performs a full backup on all the nodes. After that, it performs incremental backups three more times that day. So by the end of the day four versions of the data will be available. Future days follow this same pattern; at night we replace our four versions with a single, current version, and at three other times during the day incremental backups occur creating new versions.

3.4 How will we store our chunks?

When the system backs up chunks **A** and **B** it will create a replica of each of them. These replicas will be stored on non-origin peers. In addition to these replicas we will create redundant replicas, but save disk space in storing them by using **XOR**. Instead of storing a second copy of both **A** and **B** we would store $\mathbf{A} \oplus \mathbf{B}$.

The utility of this is that if any one of these three points fails, then it can be recreated using both of the other two. If the node storing **A** were to go down, then restoring it would be a matter of taking **B** and performing $(\mathbf{B} \oplus (\mathbf{A} \oplus \mathbf{B}))$ and we would have recovered **A**.

There are several ways we can tune this process. We can store more replicas of each chunk to reduce the likelihood of data loss. We can also XOR together more than two replicas to continue to increase redundancy while saving disk space. These are issues of preference, and we will be consulting Jeff for his opinion as well as developing algorithms to decide which chunks to replicate and which to XOR together. These algorithms will attempt to make the system as robust as possible to node failures for a given amount of disk usage.

3.5 Where will we store our chunks?

This is a matter we haven't fully figured out yet. When we wish to store chunks **A** and **B** we will select a peer based off of network topology or an algorithm that determines optimal chunk placement for maximum reliability. The distributed hash table will store some metadata about the replica, and its location. Where the metadata is stored in the DHT and which peer gets chosen are still matters of discussion.

3.6 So what happens when nodes fail?

There are two issues that arise when nodes are unreachable. First of all, the backup process generally expects all nodes to be available when the backup is run, so we need to determine what to do if this is not the case. Secondly, the purpose of the system is to allow restoring the data if a node fails, so we need to know how to collect the data that belongs to a node and restore it to the node's hard drive.

Backing up We need a procedure for backing up all the data when a node is unavailable for a backup, and a way of balancing things if a node is unavailable for the first backup of the day but joins the network for later backups. Because only metadata is stored in the distributed hash table, our system is not very picky about which node stores what data, but we still need to make a decision about what the system should do in this kind of situation.

Restoring Minimally, our system must be able to restore a full backup of a node onto the **/local** directory of that node in the event of a disk failure or similar problem. We will do this by retrieving from the DHT a list of directories on that node, getting the metadata for each of the chunks associated with those directories from the DHT, and requesting the actual chunks from the peers who, according to the metadata, have them stored away. Some of these peers may need to reconstruct data using XOR coding, which requires downloading additional data from other peers. Since this is slow, we will avoid doing it if simple replicas are available. The node being restored will build its directory structure based on the list originally taken from the DHT, and fill in the directories as it receives chunks from its peers. If we have time, we would also like to implement an interface for retrieving particular files and directories from the backup system. At first, this could be as simple as a script that could retrieve the file **/lemon/local/student1/img/puppies.jpg** from the system, but ideally we would like to be scan through the folders *a particular student* has created on the **/local** directories of the various computers so that a user can find the file that interests him even if he doesn't know the name of the computer that stored it.

3.7 What Special Features Might our System Have?

Give Up Fairness for Reliability Many implementations included some form of fairness guarantee. For example, the data a node could back up in the system might be proportional to the storage it offers to other nodes on the system. However, our goal is to back up the entire lab as reliably as possible. To this end, we will try to balance the storage of backup chunks evenly across all peers, regardless of the amount each node stores in the system. This balancing will maximize the availability of the data we need, and probably simplify the decision of where to back up a particular chunk.

User Tags A feature we have been discussing, and would only be implemented late in the proejct, is a metadata browser that would allow a user to query the system for all backups of folders that he or she had created. This would be accomplished with the use of the system's metadata at time of backup. This would be useful if a user could not remember which computer he or she had been using for its local directory.

4 Evaluation

We will evaluate our system based on its ability to restore data in different scenarios of node failure. Additionally, there are a few metrics in which we should be interested.

4.1 Algorithms

Along the way, in our system's implementation, we will be presented with opportunities to try out several algorithms to meet a certain goal. There is enormous potential for experimentation at these points. One such example is *how best to chose an XOR partner*

4.2 Failure Transparancy

We might want to see if our network can handle a node's leaving the network unexpectedly. This test can be run at various times.

1. Node down at time of backup.
2. Node down at time of restore.
3. How comprehensive a restore is available for different degrees of failure.

4.3 Time to Back up

1. How much time will it take to backup a large portion of data. (Large Folder Contents)
2. How much time will it take to backup a complex portion of data. (Many Nested folders)

5 Equipment Needed

For this probject we will require some additional access to the lab computers, and the ability to install the peer to peer infrastructure and DHT on the computers. We will require backup locations, where the replicas will be stored, on all of the lab computers. For testing we can use directories within **/local**, but if we want to implement the system for real we will at least need a **/backup** or some such directory for the system to write to.

6 Schedule

Stage 1: Network Topology Questions and Tests
Expected Time: 2-3 days

Stage 2: Implement and become familliar with a peer-to-peer infrastructure and associated DHT
Expected Time: 7 Days

Stage 3: Implement initial chunking division (IE: create the chunks, based off of the local filesystem.)
Expected Time: 3 days

Stage 4: Enable backup and restore of chunks locally, to verify our chunks are correct and we know how to use them.
Expected Time: 2 days

Stage 5: Enable backup and restore of chunks on Peer-to-Peer Network
Expected Time: 3-4 days

Stage 6: Implement tests to ensure correctness of solution
Expected Time: 1-2 days

Stage 7: Implement xor of replicas on network
Expected Time: 4-5 days

Stage 8: Implement versioning
Expected Time: 3-4 Days

Stage 9: Implement per-user metadata (single-folder restore)
Optional Extra Step–Expected Time: Whatever is Available.

7 Conclusions

We hope that our system will provide the Computer Science Department a clean and reliable backup system for their currently non-backed up **/local** directories. We will do this by implementing a peer-to-peer system, which makes use of network stored chunk replicas, with redundancy provided by space-saving XOR parity data. Metadata pointing to the locations of the chunks will be stored in a distributed hash table, while chunks will be stored on specific machines selected by algorithm, not by hash. Chunks will contain the files in directory, allowing additional utility for the system than just full-node restores. We will test the scalibility of our system through a suite of projected tests, including dropping and adding nodes. We acknowledge that this project is ambitious, so our schedule is ordered to ensure that the critical functions for a backup system will be implemented even if we run out of time.

Annotated Bibliography

[1] Christopher Batten, Kenneth Barr, Arvind Saraf, and Stanley Trepetin. pStore: A secure peer-to-peer backup system. Technical Memo MIT-LCS-TM-632, Massachusetts Institute of Technology Laboratory for Computer Science, October 2002.

> The system presented in this paper, pStore, breaks files up into blocks and stores them in a distributed hash table, implemented using Pastry. pStore uses some interesting techniques, like convergent encryption and a fast algorithm for finding duplicate data, but creates both storage and processing overhead that would be problematic for satisfying our chosen goals.

[2] Samuel Bernard and Fabrice Le Fessant. Optimizing peer-to-peer backup using lifetime estimations. In *EDBT/ICDT '09: Proceedings of the 2009 EDBT/ICDT Workshops*, pages 26–33, New York, NY, USA, 2009. ACM.

This paper discusses an heuristic for selecting peers based on the hypothesis that peers that have been part of a backup system for a longer time will be more invested in the system, and will therefore be less likely to abandon the system and the peers whose data they back up. The results presented in the paper are interesting, and suggest that this is in fact a useful heuristic for selecting peers in an internet setting. However, the results are not relevant to our work because we operate in a lab where all machines can be trusted to be available when a restore needs to be performed unless prevented by some kind of problem.

[3] Eduardo M. Colaço, Marcelo Iury S. Oliveira, Alexandro S. Soares, Francisco Brasileiro, and Dalton S. Guerrero. Using a file working set model to speed up the recovery of peer-to-peer backup systems. *SIGOPS Oper. Syst. Rev.*, 42(6):64–70, 2008.

This paper discusses a system for reducing the time required to bring a crashed node to a working state by guessing which files are most likely to be needed and restoring those first. This is a valuable feature of a backup system, but it requires a great deal of infrastructure for determining when files are used, as well as the heuristics that determine which are the most important, and neither of those were issues we were prepared to tackle. If all goes well, we implement the ability to restore individual files by filename, which could provide a form of this timesaving mechanism by allowing the user to manually restore his important files while a longer restore takes place.

[4] Landon P. Cox, Christopher D. Murray, and Brian D. Noble. Pastiche: making backup cheap and easy. *SIGOPS Oper. Syst. Rev.*, 36(SI):285–298, 2002.

This paper is absolutely full of ideas. Pastiche is a system for backing up the entire contents of a user's computer on several buddies, which are selected to have large amounts of data in common with the user's computer. Pastiche introduces several interesting ideas, including content-based indexing to find shared data more effectively, convergent encryption to protect data while being able to recognize duplicate data, and a system of finding peers based on broadcasting to random (but disjoint) subsets of its peers. Though this paper contains many interesting ideas, it is a complicated solution and mostly solves problems that don't affect our lab.

[5] Martin Landers, Han Zhang, and Kian-Lee Tan. PeerStore: Better performance by relaxing in peer-to-peer backup. In *P2P '04: Proceedings of the Fourth International Conference on Peer-to-Peer Computing*, pages 72–79, Washington, DC, USA, 2004. IEEE Computer Society.

PeerStore is in many ways a response to pStore [1]. It proposes an alternative way of using a distributed hash table, storing the metadata in the DHT and putting the file data directly on peers according to some algorithm. For this algorithm PeerStore uses symmetric trading to ensure fairness in resource utilization by peers. Fairness is not important for our system, so our algorithm will be more focused on reliability, but the idea of separating file location data (metadata) from file contents is one we used extensively.

[6] Boon Thau Loo, Anthony LaMarca, and Gaetano Borriello. Peer-To-Peer Backup for Personal Area Networks. Technical Report IRS-TR-02-015, Intel Research Seattle - University of California at Berkeley, May 2003.

This paper presents a system for backing up small devices over a Personal Area Network. It has some interesting ideas, but they are mostly related to reducing or budgeting the power consumed by the device in transmitting to other nodes. We have no such constraints, so the innovations of this paper are not very useful to us.

[7] Dinh Nguyen Tran, Frank Chiang, and Jinyang Li. Friendstore: cooperative online backup using trusted nodes. In *SocialNets '08: Proceedings of the 1st Workshop on Social Network Systems*, pages 37–42, New York, NY, USA, 2008. ACM.

Friendstore is a system where peers are selected based on social relationships between their owners. It focuses mostly on the issue of ensuring that peers continue to provide their services, for example, in case of a computer needing to do a restore. This is not an issue we need to deal with, because we control all of the computers involved. However, this paper peripherally brings up the idea of XOR coding to trade off disk space for network bandwidth, an idea we found useful for our system.