Magnus: Peer to Peer Backup System

Naveen Gattu, Richard Huang, John Lynn, Huaxia Xia

Department of Computer Science
University of California, San Diego

Abstract

Magnus is a peer-to-peer backup system for reliable and persistent data store using surplus resources. Magnus achieves its reliability and minimizes storage by using Erasure Coding to distribute fragments to different peers for backup. Decoding the fragments takes negligible time compared to retrieving data over the network and writing data to disk for recovery. Additionally, monitoring online status of peers ensures reliability by shifting fragments stored in offline hosts.

1 Introduction

Traditional file backup systems depend on reliable storage devices such as tape drives, zip drives, and CD-burners. However, costs associated with these reliable secondary storage devices as well as management of media are expensive. Alternatively, Internet backup sites offer backup services for monthly fees. Costs associated with these online backup sites are expensive as well because of the dedicated resources associated with backing up data.

We propose a peer to peer backup system based on the slack resources at each of the participating hosts. Recent trends suggest that the disk capacity of computers has skyrocketed; if growth continues according to Moore's Law, a *terabyte* of EIDI storage will cost $100 by 2006 [1]. Each participating host can receive as much backup space as it is willing to allow others to store, thus ensuring that sufficient space for all parties.

Magnus requires the participating hosts to be connected via a Local Area Network and be online normally to allow backup and restoration of files to occur at any instance. To tolerate some unexpected downtime to some of the participating hosts, we use Erasure Codes to tolerate (n – k) hosts to be simultaneously offline, where n is the number of fragments we store on peers and k is the number of fragments we need to recover the original file.

We anticipate Magnus to be initially deployed in a trusted network such as in a university or a corporation. We can extend the work to a global overlay network when we can use a system like Chord [2] distributed lookup system to locate the servers responsible for storing a fragment.

The remainder of this paper is organized as follows. Section 2 outlines our goals of designing this system. Section 3 presents the functionality of Magnus. Section 4 discusses comparison of Magnus to other backup data storage. Section 5 presents performance numbers. Section 6 discusses related work. Section 7 discusses open issues and future work.

Finally, section 8 summarizes our findings and concludes the paper.

2 Goals

When designing our system, we adhered to the following goals:

- *Reliability*. Our top priority was to ensure that our system was as reliable as a secondary storage device. We want to be able to retrieve data even when some hosts holding our data are down.
- *Storage space.* We want to minimize the amount of data stored in each of our peers. We don't want full replication for our data.
- *Speed of recovery*. Recovery from a disk failure should be dominated by the network bandwidth and disk IO speed for writing the recovered files to disk.
- *Ease of use.* Users should be able to backup a file by inputting at the command prompt without having to worry about where his files are stored and the files should be automatically recovered in the event of disk failure without the user having to specify how or where from to recover the files.
- *Cost.* Our backup system should not cost the user any additional hardware or resources other than slack resources.

3 Design Overview

Magnus provides distributed data storage and retrieval. It is structured as a collection of peers sharing excess resources for storage. Each peer can store as much data as it is willing to provide for other peers. We use Erasure Coding to break up each file the user wants to backup into *n* fragments and send each fragment to a random host from a list of known hosts. To recover a file, *k* fragments are required to construct the original file.

From the command line, the user can automatically backup a file without worrying about where the file is stored. When a power outage occurs, Magnus restores all in-memory structures from hard disk. When a disk failure occurs, presumably a system administrator repairs or replaces the disk. After installing the software for Magnus and starting the local p2pbackupd daemon, recovery of all files on that machine occurs automatically without user intervention.

For ease of use, Magnus also offer the option of recovering a single file in the event of accidental file deletion. The user can also delete a file and any associated fragments at other peers when that file is no longer needed.

3.1 Codec

The codec module provides the erasure coding mechanism for the backup of files. Files are first fragmented into K equal sized fragment files (padding added if necessary for clean division), then *N-K* parity files are computed and output with the original *K* fragment files. The codec performs erasure coding using Vandermonde matrices[1]. An (*N-K*) x *N* Vandermonde matrix V is defined as: $v_{i,j} = j^{i-1}$. And an array A= $\begin{bmatrix} I \\ V \end{bmatrix}$, (I is the *N x N* identity matrix) has the property that any subset of K rows is linearly independent. Thus, by matrix

[1] The byte level coding/decoding routines are provided by a forward error correction package developed and open sourced by Luigi Rizzo [3].

multiplication we may encode a *K* fragmented file into N encoded fragment files, from which we may reconstruct the original *K* fragmented file in the event of *N*-*K* encoded fragment loss. Information for the location of these fragments among the peers is stored in the FileMap, and Peer selection is done using the PeerManager function ; GetRandom peers for each fragment. During the recovery of a file, the codec is invoked by the Recovery Manager with a subset of the *K* encoded fragments. Note, that the encoding is systematic, meaning that the first *K* encoded fragments are identical to the original *K* file fragments. Therefore if the first *K* fragments are received, no decoding overhead is incurred. Decoding is performed with Gaussian elimination.

3.2 FileMap

FileMap is constructed as a linked list of entries. Each entry has the unique key of filename and version number. Additionally, each entry contains *n* and *k* for that file, the information about the hosts where each fragment is stored. To facilitate updating entry information on disk, FileMap keeps a vector of positions corresponding to each entry on disk. An additional list contains the deleted entries. When a new entry is written to disk, FileMap first checks to see if any deleted entries exist and allows the new entry to go into that slot on disk.

3.3 FileDatabase

The FileDatabase maintains a record of backup file fragments received from peers. It supports queries from a recovering host to return file header information to allow the recovering host to recover the most recent versions of the files and also to construct the FileMap. In order to protect this information during a local power failure, an mmap is used to synchronize FileDatabase information with stable storage. This method provides a very fast and efficient mechanism for disk synchronization and allows a rapid recovery of data during the startup of the backup daemon.

3.4 Peer Manager

The Peer Manager provides an abstraction for accessing peers in the Magnus network. It provides mechanisms for obtaining references to and enumerating through peers that are currently active. The purpose of the Peer Manager component is to provide a simple interface for locating peers on which to store file fragments and also as a simple way of determining whether or not a peer is currently active.

An important design consideration for the Peer Manager is how it enumerates through active peers. If a network failure were to cause a set of collocated peers to become unavailable for a period of time then any file whose fragments are distributed among those peers will have an increased chance of being unrecoverable. It is important then that the Peer Manager not enumerate sequentially peers that can be affected by the same physical failures.

In the initial implementation of Magnus the Peer Manager maintains a flat list of peers in order to provide this functionality. However in the interests of scalability we envision this component to utilize a system such as Chord to provide access to global-sized networks. The issue of enumerating through peers in such a way that reliability is improved was not addressed. Instead we chose to use

3.5 Peer Interface

The Peer interface provides a reference to remote instances of Magnus servers. All method invocations on remote servers are asynchronous RPC-like calls on Peer objects. A Peer object exposes four methods for interacting with remote servers:

- **StoreFile(file, cb) –** Sends a file to be stored on the remote server.
- **RetrieveFile(file, cb) –** Requests that the peer return the specified file.
- **RemoveFile(file, cb) –** Tells peer to remove the file from its FileDatabase.
- **QueryFiles(file, cb) –** Asks peer to return a list of files whose metadata matches the metadata in file.

Each asynchronous method also takes as a parameter an object that implements the RemoteServiceCallback interface. Once the remote operation has completed, the callback object is notified with the result and a status variable that reflects the success of the call.

Magnus takes advantage of asynchronous calls by storing and retrieving multiple files at a time. This allows us to obtain better performance by transferring files in parallel rather than sequentially.

3.6 Recovery Manager

The Recovery Manager is invoked when the user wishes to recover a single file or when the system recovers from a disk failure. It consults the FileMap to get information about which peers to contact, initiates connection to those peers and calls Codec to reconstruct a file whenever k fragments are returned.

3.7 Reliability Monitor

At regular intervals, the Reliability Monitor sends out probing messages to the peers storing backup data. For any peer that does not reply within a certain timeout, that peer is declared down and the Reliability Monitor informs the Peer Manager and updates the FileMap to reflect the down status of that peer. Magnus can tolerate (n - k) simultaneous peers down; when (n – k) peers storing fragments for a file is declared down for a particular file, the Reliability Monitor redistribute those fragments to new peers and updates the FileMap. When a peer comes back online whose fragments has been shifted by the Reliability Monitor, the Reliability Monitor will send a message to that host to delete the fragments that has been shifted.

4 Discussion

Magnus provides reliable, persistent backup. We compare Magnus to secondary storage devices and full replication at peers in terms of reliability, storage considerations, speed of recovery, ease of use, and costs.

4.1 Secondary Storage Systems

Secondary storage systems such as tape drives, zip drives, and CD-burners are very reliable. Magnus achieves similar reliability by monitoring the online status of peers and can tolerate (n - k) peer failures and simultaneous failure by the original host before the Reliability Monitor can act.

Costs incurred by using secondary storage include new hardware purchase, management of media, and transportation costs for media if users wanted to keep data in geographically separate locations. Magnus does not require any special hardware, but simply uses slack resources.

In terms of usability, Magnus allows backup to process in the background whereas zip drives or CD-burners require users to wait for the media. The time Magnus takes to recover from a disk crash is dominated by the network speed. The trend towards high bandwidth networks bodes well for the success for Magnus. Random accesses to recover a single file or to delete a file are strong advantages over secondary storage devices.

4.2 Full replication

To take advantage of slack resources at peers, another scheme is to fully replicate data at peers. The reliability of such a scheme is as good as Magnus when the number of copies equals k. However, the Reliability Monitor within Magnus will shift fragments when peers are detected to be down, thus ensuring the reliability of data dynamically. No such assurances exist for a full replication scheme without a Reliability Monitor.

When recovering from a disk crash, the user will not have the benefit of the Peer Manager and therefore would need to manually store the location of peers to manually retrieve lost data.

5 Performance

We tested and evaluated Magnus on the UCSD Active Web machines and presented our findings below.

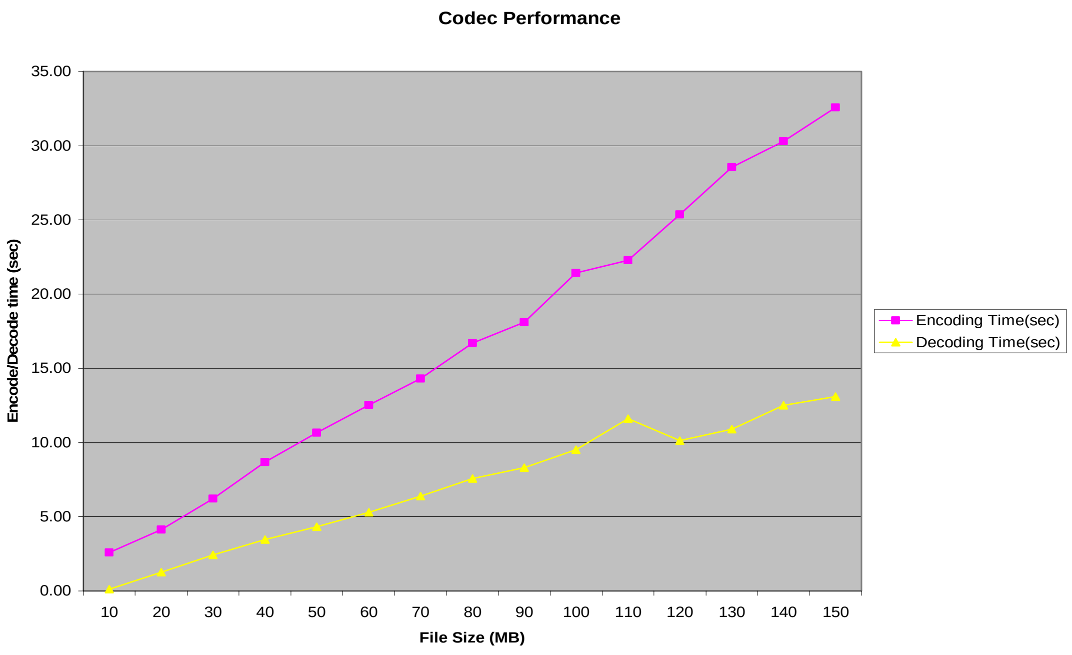

Figure 1

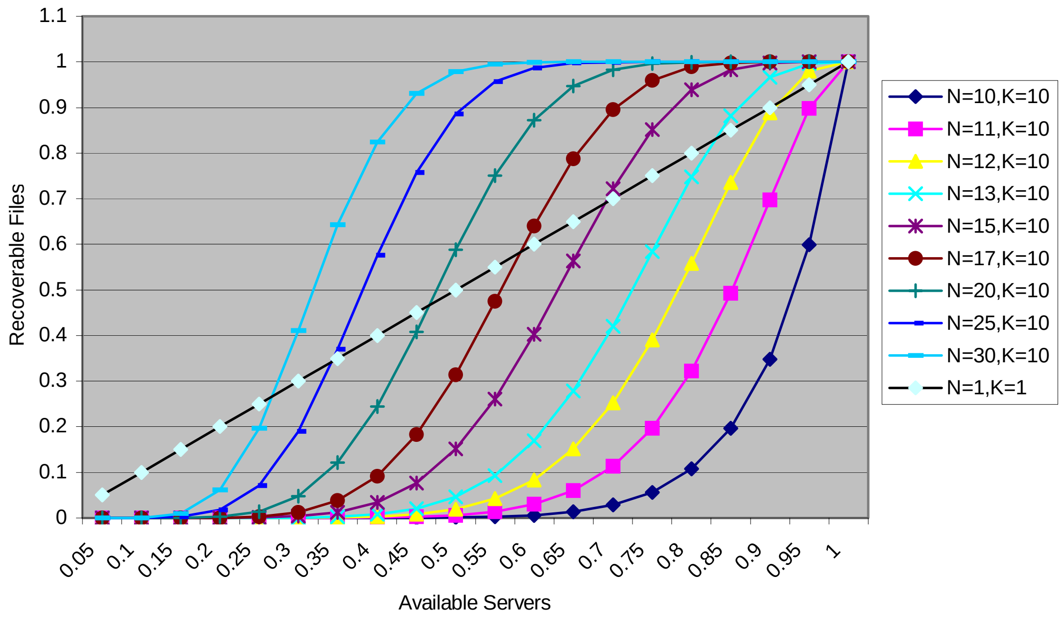

Figure 2. Reliability Based-On Erasure Code

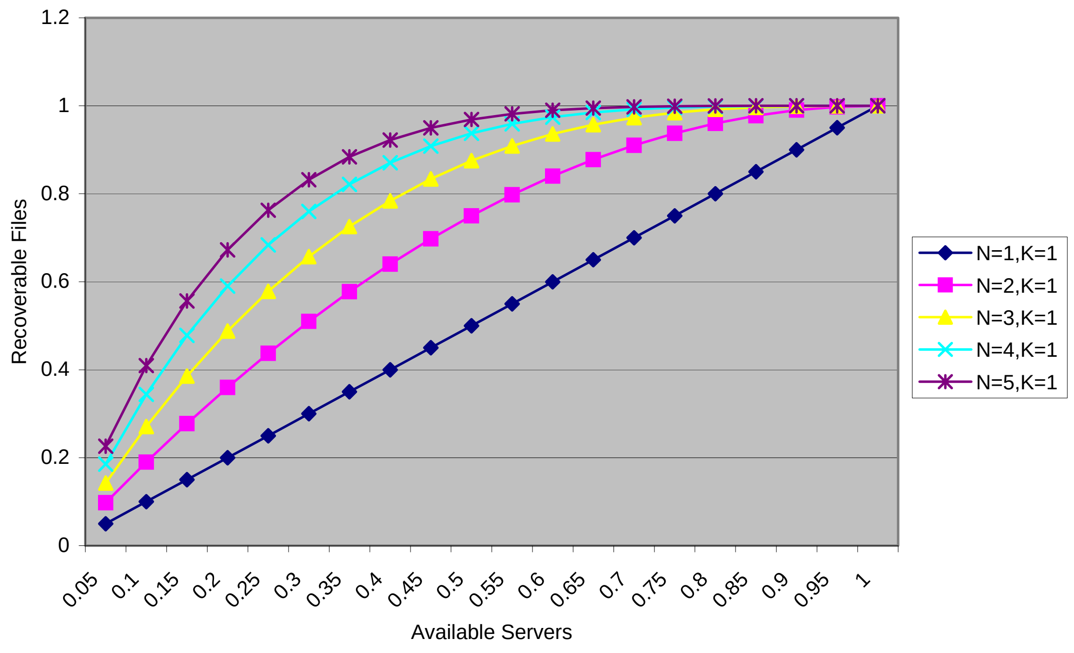

Figure 3. Reliablity based-on Simple Replication

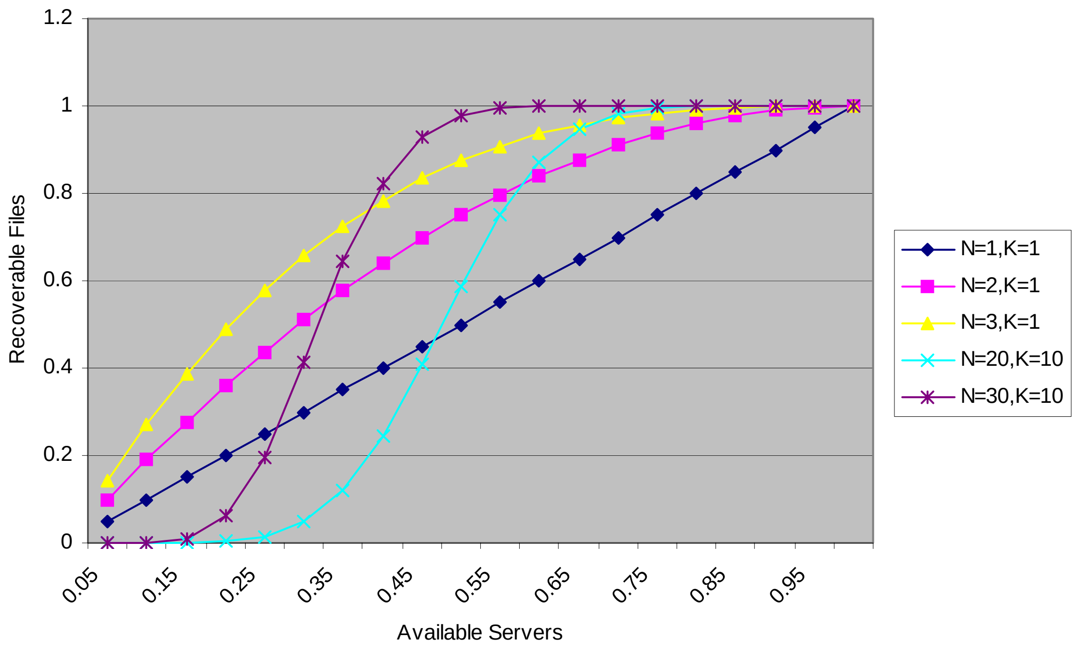

Figure 4. Comparison between replication and errasure code

5.1 Codec Performance

We first test the encoding/decoding time on dual PIII 800MHz machine with FreeBSD 4.6.2.

In figure 1, we show the performance for encoding fragments with *n* = 10 and *k* = 5. We plotted the encoding/decoding times as a function of the file size. We noticed a linear relationship between file size and encoding/decoding times.

The encoding bandwidth is about 4.6MBps; and the decoding bandwidth is about 11.5MBps. By showing this, we argue that the file re-construction bandwidth is higher than the network bandwidth for most of distributed systems.

5.2 Optimal Fragment Number

Reliability is our major concern. We analyze the reliability against different values of N, K and server failures (1-S).

In the analysis, we assume that the total number of servers is much larger than N. Therefore, given N, K and S, the rate of recoverable files is:

$$\sum_{i=0}^{N-K} \binom{N}{i} S^{N-i}(1-S)^{i}$$

In figure 2, we show the impact of picking different *n* values for a fixed number of *k* (10). When (*n – k)* increases (with higher values of *n* and fixed values of *k*), the probability of a file recoverable increases in the face of offline peers. In the case that a majority of the nodes are alive, erasure coding provides a very good reliability to storage consumption ratio. For example, with N=20 and K=10, 99.6% of the files are recoverable when 25% of the total servers fail; 94.7% of the files are recoverable when 35% servers fail. With N=30 and K=10, at a 30% failure rate we achieve a 99.99% recovery probability and 97.9% on a 50% total server failure event.

As a comparison, we also show the performance of pure file replication in figure 3. Using five times of the storage space required for a single file, pure replication achieves a 99.99% recovery probability for a 10% server failure . In figure 4, we compares (N=30, K=10) against (N=3, K=1), and (N=20, K=10) against (N=2, K=1). This shows that using an equal amount of storage, erasure coding provides a much higher reliability when server failure rate is less than 50%.

6 Related Work

Magnus was inspired by the abundance in peer-to-peer systems today such as Gnutella [4] and Kazaa [5]. The skyrocketing capacity of disk storage and the corresponding decrease in cost has led to very natural choice to consider storing data on slack resources at peers.

The Cooperative File System (CFS) is a peer-to-peer read-only storage system that provides provable guarantees for the efficiency, robustness, and load-balance of file storage and retrieval [6]. The basic unit of storage in CFS is a block. CFS provides a distributed hash table using Chord for block storage.

OceanStore is a utility infrastructure designed to span the globe and provide continuous access to persistent information [7]. The authors of OceanStore envisioned a utility model in which consumers pay a monthly fee in exchange for access to persistent storage. This is not dissimilar to Internet backup sites offering reliable storage of data. It’s not clear why any user would want to store their data globally, although such a system guarantees persistent data in the face of catastrophe to continents.

7 Future Work

As this paper describes both the designs and the work in progress of Magnus, some components have yet to be fully implemented. One glaring component missing is the Reliability Monitor. Without the Reliability Monitor, Magnus still has the basic functionalities of backing up a file to peers and tolerating (n - k) offline peers, but it cannot dynamically shift the fragments to other online peers.

The Peer Manager as currently implemented is naïve in choosing a random peer to send a fragment. It also requires knowledge of all peers at start-up, thus not very scalable. Using Chord or some other underlying host lookup system would improve performance during recovery and possibly provide better load-balancing among peers. In addition, a dynamic host recognition system would be more scalable.

Magnus currently allows different versions of a file to be backed up at peers. Each version is treated as a separate file. Future research would be to store only incremental changes between different versions.

We presented the implementation of Magnus without addressing much of policy. Such issues as cheating servers and malicious servers need to be addressed. Security issues in regards to data privacy can be solved using different forms of encryption before sending to peers.

8 Conclusions

Magnus is a highly reliable, cost-effective peer-to-peer backup system using surplus resources. It allows user reliable storage while minimizing data storage at other peers. We achieved near FTP speeds during recovery while greatly reducing user intervention in recovering a file from peers. The cost of the system is in the surplus disk space at each participating peer.

A prototype of Magnus has been implemented and evaluated in the UCSD Active Web cluster. Future deployment over wider area networks will likely uncover opportunities for improvement, but the current results indicate that storing files at peer hosts is a viable option for a backup system.

References

[1] S. Rhea et al. Pond: the OceanStore Prototype. In *Proc of the 2nd USENIX Conference on File and Storage Technologies*, 2003.

[2] **J. S. Plank. *A tutorial on Reed-Solomon coding for faulttolerance in RAID-like systems*. Software---Practice and Experience (SPE), 27(9):995--1012, Sept. 1997.**

[2] I. Stoica, R. Morris, D. Karger, F. Kaashoek, H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *Proc. ACM SIGCOMM*, San Diego 2001.

[3] L. Rizzo. Effective Erasure Codes for Reliable Computer Communication Protocols. *ACM Computer Communication Review., vol 27(2),* Apr 1997, 24-36

[4] Gnutella website. http://gnutella.wego.com

[5] Kazaa website. http://www.kazaa.com

[6] F. Dabek, M. F. Kaashoek, D. Karger, R. Morris, and I. Stoica. Wide-area cooperative storage with CFS. In *18th ACM Symposium on Operating Systems Principles (SOSP '01)*, October 2001, pp. 202–215.

[7] J. Kubiatowicz et al. Oceanstore: An architecture for Global-Scale Persistent Storage. In *17th ACM Symposium on Operating Systems Principles (SOSP '00)*, November 2000, pp. 190–201.