Project Mimir: Breaking from Replication Based Storage for Large Scale Dynamic Systems

Ryan McGovern
College of Engineering
University of Cincinnati
Cincinnati, Ohio 45220
Email: ryan.mcgovern@lerao.com

Chris Siebert
College of Engineering
University of Cincinnati
Cincinnati, Ohio 45220
Email: sieberca@gmail.com

***Abstract*—Demand for large reliable data storage is increasing as single disk reliability is decreasing. SAN storage solutions are expensive, difficult to setup and maintain, and are difficult to expand. We propose Mimir, a distributed file storage system utilizing erasure codes over a distributed hash table based peer-to-peer network of computers. By pooling a large number of multi-purpose computers over a LAN, extremely large amounts of storage space can be utilized in a largely decentralized but secure network. Parallel write and read operations enable high network throughput performance while erasure codes allow the system to tolerate high variance of disk performance and reliability. By using erasure codes, the reliability and disk utilization efficiency can greatly exceed that of similar replication based systems with comparable speed performance. The high redundancy of erasure encoding and network flexibility of a distributed hash table creates the potential for Mimir to maintain a stable file system over an unreliable ́high churnńetwork. Already tested on networks of greater than fifty computers, file integrity is maintained with network losses greater than fifty percent. The abstract goes here.**

I. Introduction

The purpose of our project is to solve a very specific problem in the field of large scale storage. This field has had many different attempts at solutions. In our opinion their are three areas the prevailing solutions do not handle well; scalability, cost, speed or desired use.

The first area is in scalability; not just in how big you can make one but how well you can add storage to a system. Systems that do this well are OceanStore, RobuSTore, GFS, iDIBS. However none of these fully satisfy the requirement [1] [2] [3]. OceanStore has needed a large team of researchers to attempt to get a high level of scaliblity and in other replication based systems they do not scale to such large sizes well. GFS is probably the most successful of these listed but however it is not available to anyone to actually use and test so it is not a viable solution for those companies that need an in house storage solution and can not outsource that responsibility to google. It also still requires off site backups because it is not robust for slow links. iDIBS is an extension of the DIBS project which does a good job of providing distributed backups and is very scalable allowing backups to be distributed accost the entire internet however it comes at the cost of reliability, control and speed. The closest project to ours is RobuSTore which uses the same erasure codes as us but is built upon a somewhat static network model that requires high speed links and it is optimized for large file storage and not designed for general business use cases.

The second area is in cost; with personal computers coming with larger drives it is not as necessary to have large centralized systems with lots of wasted space on a users personal system. Systems that may fulfill other areas but do not fulfill the requirement of low cost are systems such as Storage Area Networks and Network Attached Storage both of which require large and expensive infrastructure especially considering the cost of running fiber and copper cables for users to connect to these systems. While GFS does work well with commodity hardware the machines are still dedicated to the purpose of storage.

The final issue is in goal usage; it must be usable for real time document storage and retrieval not just as a backup system. A good distributed storage system should not need to be backed in case of failure. It should provide high enough levels of reliability as to satisfy at-least non temporal backups. Also it must be fast enough to be used for real time data access. Systems like iDIBS and LanStore are designed as backup systems instead of storage systems [4] [5].
With these problems in mind we created a system that is dynamic relying very little on any central server that uses storage nodes running on peoples personal computers and is fast enough and reliable enough to fulfill both roles of real time document storage and backup.

II. Mimir's System Architecture

In this section we will discuss the general system architecture that was developed to allow our dynamic storage system while providing good latency and throughput.

A. Overview

In our system there are three types of nodes; a Metadata Controller node (MDC), Storage nodes, and Client nodes. The Metadata controller provides a cryptographic key center as well as retaining information about the state of the distributed storage system. Storage nodes provide access to the underlying persistent storage of the machines in which they run. Client nodes provide access to the distributed data for both reading and writing to the end users. Any combination of these nodes can run on any machine, however in the current system we assume there is only one Metadata Controller node in the network. Also it does not make much sense to have multiple storage nodes for a machine. These nodes communicate using FreePastry and Scribe which are Distributed Hash Tables as well as some out of band communication that allows for increased data storage speeds.

B. Message Routing and Resource Discovery

As our goal is to be highly dynamic we decided to use a DHT based routing table and decided upon using FreePastry and Scribe. The advantages of such a system is that it allows the network to not need a central server or starting point for communications creating a bottleneck. DHT's are also better able to handle churn and can potentially solve the net-split problem. Statically defined or centralized network schemes have the disadvantage of more extensive setup and configuration which would not be conducive to the large scale deployment of a distributed system. To increase the dynamic nature of our system we used Scribe which is built upon FreePastry and provides multicast and anycast abilities. We use multicast to discover what storage nodes are available to receive files, as well as to request a file from the storage nodes. This is done by setting up topics for each resource. In our case our resources are storage, and metadata. Thus when a storage node starts up it joins the storage topic, and Metadata Controllers join the metadata topic. Then to contact all the subscribers of a topic we send a multicast message or to contact one node we send an anycast message.

C. Data Requests

All requests and commands are performed using this topic based system with the exception of actually sending the encoded file data to a storage node. For speed reasons when we wish to save a file we send it directly to the storage node. To implement this we maintain a cache of the current storage nodes that is kept up to date by periodically sending a multicast messages to the storage topic. The storage nodes who receive this message then send a response through pastry to the client with an IP address and port for their direct connection interface. This cache is then used to get the hosts to stripe the encoded data onto using raw binary data instead of Java Serialization as all of pastry does.

Before a client can save or retrieve a file the user must first get a id from a MDC for the file. For saving a file the mdc creates a new entry in a database, and for file retrieval it looks up the requested id for the given file. In our implementation this id done with MySQL.

When retrieving a file the data obviously comes back out of order but fortunately the decoder handles this very well since it uses the Luby Transform it does not matter the order at all for rebuilding. This allows the system to be robust by not being limited by the slowest connection. This could lead to issues with over-consumption of bandwidth but to minimize this once the file is rebuilt we tell all the storage nodes to stop sending. this would help limit the wasted bandwidth especially on slow or low bandwidth connections.

III. Erasure Codes

A. Reasoning behind Erasure Codes and Distributed Storage

Erasure codes enable a distributed storage system to eliminate the high organizational costs and fragility of similar replication based distributed storage solutions. Because replication can never be more reliable than the number of copies in storage, the location and status of all specific data must be constantly known and monitored. Erasure codes, which encode a file into $n + m$ blocks (where n is the number of original blocks and m is the number of encoded blocks) require only that the total number of encoded blocks are present in the system. No specific encoded blocks must be monitored, only that at least fn blocks, where ≤ 1 are available in a relatively even distribution is required to ensure data reliability. It is the lack of structural organization which makes erasure codes ideally suited for decentralized storage.

One major drawback of most erasure codes is that to maintain the erasure code as encoded blocks are lost, the entire file must be reconstructed and re-encoded to the network. All previously encoded blocks are useless after this maintenance. This is because nearly all erasure codes have a rate $R = \frac{fn}{(n+m)}$. Because erasure codes such as Reed-Solomon and traditional LDPC codes have a set rate, they are only suitable for low write systems little network churn.

Rateless erasure codes were first published in 2001 by Michael Luby when he discovered the Luby Transform [**?**]. Rateless erasure codes can generate a virtually infinite stream of encoded data that is non-specific to other encoded data. The encoding is not predetermined but randomly created according to a degree distribution. An encoded block's degree is the number of blocks which are encoded together by the XOR operation.

B. Optimizing Rateless Erasure Codes for LAN

Rateless erasure codes used on a LAN, even with fairly unreliable commodity hardware, is relatively stable compared to WAN. A LAN is also generally well connected with high bandwidth throughput. To take advantage of bandwidth, encoding and decoding time must be minimized by decreasing n and m proportionally. Encoding time increases exponentially with respect to n and m with the advantage of increasing the efficiency of f.

C. Rateless Erasure Codes and Persistent Storage

Using rateless erasure codes for persistent storage requires that the erasure code is not truly rateless anymore, because all encoded blocks are stored in a static state the rate of the encoding can be determined. As a now pseudo-rateless code we have the opportunity to optimize the decoding process. If no losses have occurred, the optimal decoding speed can be achieved by receiving n distinct blocks of degree 1, which requires no decoding. To increase decoding speed we encode each original block into persistent storage.

Rateless erasure codes decode probabilistically; decoding successfully with reasonably high certainty within close proximity to fn. The requirements for persistent storage require a much higher certainty than those afforded by the most efficient rateless erasure codes. The performance and efficiency of rateless erasure codes such as the Luby Transform or Raptor Codes is dependent upon its degree distribution. The Robust Soliton Distribution developed by Michael Luby is optimized for minimizing f in the average case. But for persistent storage, the degree distribution needs to minimize f in the worst case.

Rateless erasure codes are low density erasure codes in that the average degree of an encoded block is generally only a small percentage of n. Low degree encoded blocks decode quickly with high probability while less frequent high degree encoded blocks fill in missing blocks should they decode successfully. By increasing the frequency of high degree encoded blocks the code loses average efficiency but increases its probability of successfully decoding the entire file. The difficulty is in minimizing increased inefficiency while increasing the high degree encoded blocks.

D. Adaptations of the Robust Soliton Distribution

Because we encode n original file blocks, much more than the Robust Soliton Distribution, blocks of degree two become less beneficial. The abundance of original blocks allows for blocks with a degree of three to be more easily decoded. Our first adaptation is to uniformly shift the degree distribution to one degree higher, eliminating all blocks of degree two.

In order to optimize f in the worst case we have developed a few modifications on the Robust Soliton Distribution. Higher degree encoded blocks lower the upper bound of f at a cost to the average f. Uniformly increasing the probability of higher degrees is too inefficient to be beneficial. The accumulated inefficiency as higher degrees are reached makes the performance unfeasible. To increase the higher degree encoded blocks we create a śpikeín the distribution at two points, the first and smaller spike has the purpose of keeping f at a tolerable level, the second and larger spike drastically reduces worst case levels of f.

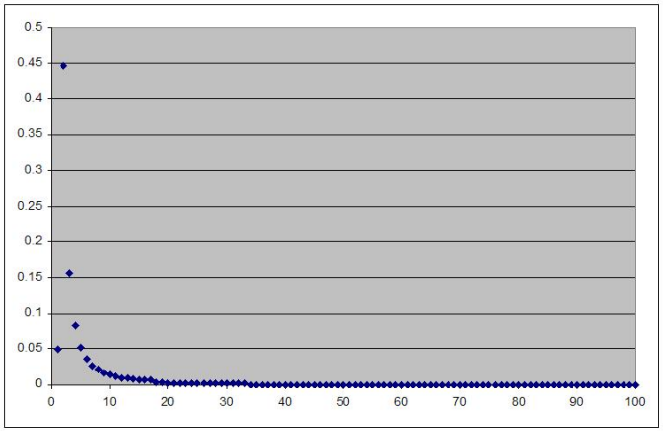

Fig. 1. Example Robust Soliton Distribution

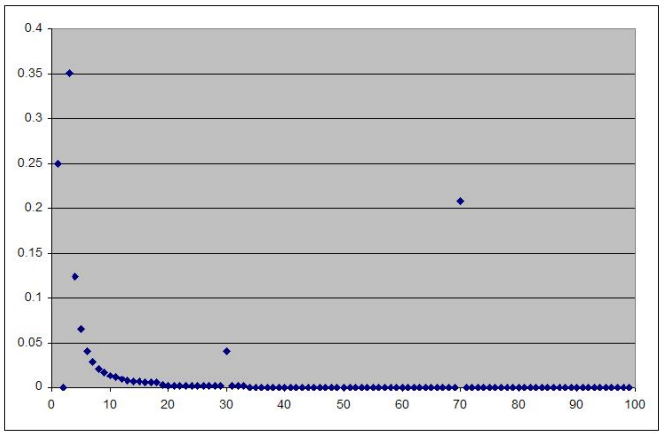

Fig. 2. Example modified distribution (note the two spikes)

IV. Results

A. Erasure Code Performance

The results of using the modified distribution resulted in a relatively minor increase in the average f from 1.25 to 1.36. Worst case f was greatly improved from a 98% to 99.9998% probability that $f2$. This means that as files can be recovered from the system with only $2n$ encoded data available with 99.9998% certainty. If $n+m$ equals $4n$ at the time of encoding, half of the network can be lost without compromising file integrity.

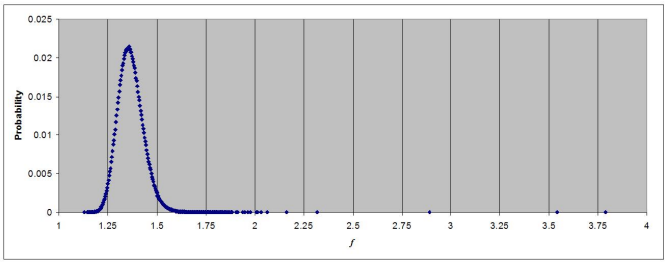

Fig. 3. Shows the probability of f, cases greater than 1.5 are unlikely

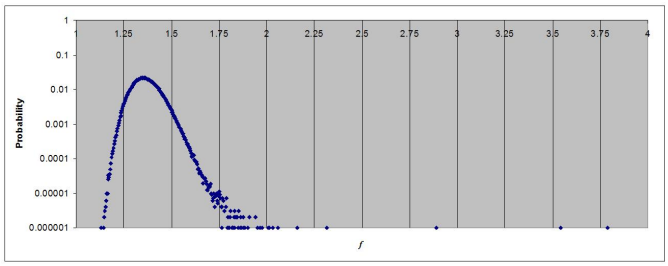

Fig. 4. Same as figure Figure 3 but with the probability shown logarithmically

B. System Performance

Limited testing has been done for system performance and we hope to perform more tests but here is the data we have. In a system with two storage nodes and a gigabit ethernet connection we received the following results. For saving and encoding a 100MB file it took 26.9988 seconds on average with a standard deviation of 2.987 seconds giving us an average throughput of 29.6Mbits/s per second output. To receive and decode the same file we took 11.73S on average with a standard deviation of 2.099S giving us an average throughput of 68Mbits/s. We also were able to run tests on UC's network using many computers in buildings across campus. The buildings are connected by two 2Gb/s fiber connections and the connections inside the buildings are 100Mb/s the results are in Figure 5.

File Size	Encode Time (S)	Decode Time(S)
200K	2.291	0.2
512K	2.42122	0.21855
1M	2.288	0.2978308
10M	4.77444496	1.5489

Fig. 5. 50 nodes on UCs Network

V. Future Work

In the future we hope to implement further improvements to speed and scalability as well as the ability to handle churn.

A. Handling Churn

By handling churn we mean the ability to handle the loss of storage nodes by rebuilding data on the fly. To handle churn we be able to determine the system health of the encoding. This task is relatively easy. By polling storage nodes for the number of encoded blocks in the system we can determine whether or not it requires rebuilding. Which data each encoded block represents is not required.

The second step to handling churn is to encode new data to bring the system level above a certain threshold. The standard method is to reconstruct the original file which can then be used to create new encoded data to a satisfactory amount. There is no need to remove previous encoded data. The problem with this method is that it requires tremendous bandwidth and computational power to reconstruct all of the files stored on the system at once. We propose to explore a new alternative way of re-encoding data once it has been stored on the network. Because the erasure code is rateless, any storage node on the network has the capability of generating new headers of encoded data for any file without knowledge of existing encoded data. The tricky part is in acquiring the data blocks required for encoding the newly generated header. Because the erasure code has a low density, only a few data blocks are required to encode an individual block. We plan to explore methods of exploiting this property to encode new data on the fly without reconstructing the original file, something which non-rateless erasure codes cannot do.

B. Scalability and Speed Improvements

One bottleneck in our current system is the number of connections that are opened when saving a file on a large network. Through some testing we have determined that this is the primary bottleneck. To solve this problem we are looking at ways to implement a more lazy method of file distribution as well as distributed encoding. In the first case we are looking at ways to just route data to end peers only checking to see if it got there by checking to see if enough data is on the network. This is as opposed to checking to see if each piece got there. though this would not help our bandwidth usage on the node sending the data it would help eliminate the large number of connections that would be made. Another method of distribution we are looking at is using a distributed encoding method. We are looking at different network graphs that would allow us to efficiently offload large amounts of the encoding onto other peers while limiting the amount of bandwidth overhead in such a system. Currently we are looking at using hypercubes but not sure where that will lead.

Acknowledgment

- Our advisor Dr. Annexstein.
- Paul Schwab and Steven Crump of UCIT for letting us use their computers for testing as well ass a sounding board for the original discussions of this project.
- Kathryn Tucker for her support.

References

[1] A. Xia, "Robustore: A distributed storage architecture with robust and high performance," *Proceedings of the International Conference for High ...*, Jan 2007. [Online]. Available: http://www-csag.ucsd.edu/papers/Xia$_s c2007_f inal.pdf$

[2] S. Ghemawat, H. Gobioff, and S. Leung, "The google file system," *ACM SIGOPS Operating Systems Review*, Jan 2003. [Online]. Available: http://portal.acm.org/citation.cfm?id=1165389.945450

[3] H. Xia and A. Chien, "Robustore: Robust performance for distributed storage systems," *Proceedings of the 14th NASA Goddard-23rd IEEE Conference on ...*, Jan 2006. [Online]. Available: http://www-csag.ucsd.edu/papers/RobuSTore$_2$0051014.*pdf*

[4] F. Morcos, T. Chantem, P. Little, T. Gasiba, and D. Thain, "idibs: an improved distributed backup system," *Parallel and Distributed Systems, 2006. ICPADS 2006. 12th International Conference on*, vol. 1, p. 10, Jun 2006. [Online]. Available: http://ieeexplore.ieee.org/search/srchabstract.jsp?arnumber=1655649isnumber=34696punumb

[5] V. Bilicki, "Lanstore: a highly distributed reliable file storage system," *.NET Technologies 2005*. [Online]. Available: