

Glacier: Highly Durable, Decentralized Storage Despite Massive Correlated Failures

Andreas Haeberlen, Alan Mislove, Peter Druschel

Department of Computer Science
Rice University

2005.08.08

Outline

- 1 Introduction
 - Motivation and Goals
 - Related Work
- 2 System Design
 - Assumptions and Failure Model
 - Design Details
 - Durability
 - Security
- 3 Experiments
 - ePOST
 - Simulation

Outline

- 1 Introduction
 - Motivation and Goals
 - Related Work
- 2 System Design
 - Assumptions and Failure Model
 - Design Details
 - Durability
 - Security
- 3 Experiments
 - ePOST
 - Simulation

Motivation

- Decentralized storage
- Correlated failures
- Tolerating k independent failures is easy, but unrealistic.

Goals

- Byzantine failures

Definition

The **durability** of a data object is the probability that it will survive a worst-case system failure.

- Aim for 6 nine's durable storage with up to 60% correlated failures.
- Willing to accept high cost (up to 11-fold overhead).

Goals

- Byzantine failures

Definition

The **durability** of a data object is the probability that it will survive a worst-case system failure.

- Aim for 6 nine's durable storage with up to 60% correlated failures.
- Willing to accept high cost (up to 11-fold overhead).

Goals

- Byzantine failures

Definition

The **durability** of a data object is the probability that it will survive a worst-case system failure.

- Aim for 6 nine's durable storage with up to 60% correlated failures.
- Willing to accept high cost (up to 11-fold overhead).

Contributions

- Efficient space and time overhead (eg. use of erasure coding)
- Provide hard durability guarantees in a hostile environment

Outline

- 1 Introduction
 - Motivation and Goals
 - Related Work
- 2 System Design
 - Assumptions and Failure Model
 - Design Details
 - Durability
 - Security
- 3 Experiments
 - ePOST
 - Simulation

Introspection

Definition

Introspection defends against the threat of correlated failures by observing actual failures to infer correlation.

- Observation does not easily reveal low-incidence failures
- Vulnerable to certain types of attacks

What's New. What's Not.

- TotalRecall targets churn, but no worst-case guarantees.
- OceanStore does not sustain massive Byzantine failures.
- Simple replication across multiple sites is old news.
- Using erasure coding for efficiency is somewhat newer.

Outline

- 1 Introduction
 - Motivation and Goals
 - Related Work
- 2 System Design
 - Assumptions and Failure Model
 - Design Details
 - Durability
 - Security
- 3 Experiments
 - ePOST
 - Simulation

Assumptions and Failure Model

- Lifetime version session time
- Most systems are online most of the time

Failure modes (implicit):

- Normal
- Failure
- Recovery

Outline

- 1 Introduction
 - Motivation and Goals
 - Related Work
- 2 System Design
 - Assumptions and Failure Model
 - **Design Details**
 - Durability
 - Security
- 3 Experiments
 - ePOST
 - Simulation

Basics

- Built on a DHT
- Pseudo-random node-id prevents Sybil attacks
- Periodic communication to detect and correct data loss
- Assumes high-speed, local **primary store** alongside Glacier.
- Fragment placement and aggregation

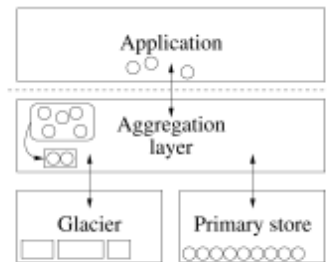


Figure 1. Structure of a multi-tier system with Glacier and an additional aggregation layer.

Basics

- Built on a DHT
- Pseudo-random node-id prevents Sybil attacks
- Periodic communication to detect and correct data loss
- Assumes high-speed, local **primary store** alongside Glacier.
- Fragment placement and aggregation

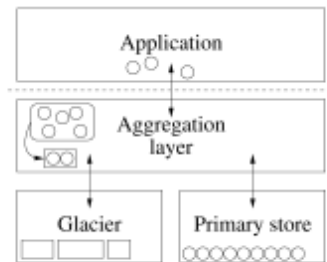


Figure 1. Structure of a multi-tier system with Glacier and an additional aggregation layer.

Interface

- Items are immutable
- Each insertion has a version number

- Insert: $\text{PUT}(I, V, O, L)$
- Retrieve: $\text{GET}(I, V) \rightarrow O$
- Refresh: $\text{REFRESH}(I, V, L)$

Fragment Details

- Fragments are created using erasure codes over the original data object.
- A hash of the fragment and entire object is stored alongside each fragment.
- Hash useful for recovery and security.
- Fragments are ideally distributed evenly across the DHT

Recovery

- Periodic (eg. hourly) update messages sent to subset of peers.
- Updates contain a list of keys stored on the node.
- Any unknown or corrupt fragments can be recovered from peer.
- Bloom filter to reduce network impact \implies probabilistic recovery.
- Garbage collection of expired fragments.
- One leak link in system is setting too short of a lease.

Recovery

- Periodic (eg. hourly) update messages sent to subset of peers.
- Updates contain a list of keys stored on the node.
- Any unknown or corrupt fragments can be recovered from peer.
- Bloom filter to reduce network impact \implies probabilistic recovery.
- Garbage collection of expired fragments.
- One leak link in system is setting too short of a lease.

Outline

- 1 Introduction
 - Motivation and Goals
 - Related Work
- 2 System Design
 - Assumptions and Failure Model
 - Design Details
 - **Durability**
 - Security
- 3 Experiments
 - ePOST
 - Simulation

Configuration

Failure f_{max}	Durability D	Code r	Fragments N	Storage S
0.30	0.99999	3	13	4.33
0.50	0.99999	4	29	7.25
0.60	0.999999	5	48	9.60
0.70	0.999999	5	68	13.60
0.85	0.999999	5	149	29.80
0.63	0.999999	1	30	30.00

Table 1. Example configurations for Glacier. For comparison, a configuration with simple replication ($r=1$) is included.

- Tradeoff between f_{max} and storage overhead
- Choosing a low f_{max} does not necessarily mean catastrophic failure.

Outline

- 1 Introduction
 - Motivation and Goals
 - Related Work
- 2 System Design
 - Assumptions and Failure Model
 - Design Details
 - Durability
 - **Security**
- 3 Experiments
 - ePOST
 - Simulation

Security

- **Integrity:** Immutability + hashes
- **Durability:** Would require targeted attack on specific nodes
- **Time source:** Use of internal timestamps.
- **Space-filling:** Does not harm existing data.
- **Glacier:** Few code paths which delete code.

Outline

- 1 Introduction
 - Motivation and Goals
 - Related Work
- 2 System Design
 - Assumptions and Failure Model
 - Design Details
 - Durability
 - Security
- 3 Experiments
 - ePOST
 - Simulation

Setup

- ePost is a “server-less” email system.
- 35 nodes
- $f_{max} = 60\%$, $P_{min} = 0.999999$
- \implies fragments $N = 48$, rate $r = 5$

System Load and Object Size

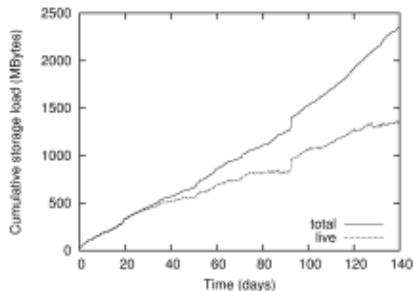


Figure 7. Storage load in ePOST.

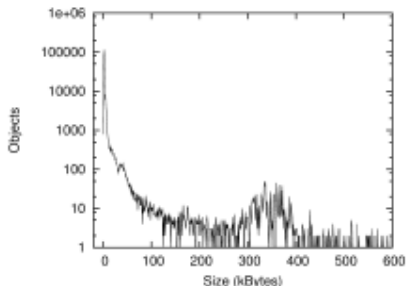


Figure 8. Object sizes in ePOST.

- Only 1% of objects larger than 600kB

Storage Overhead

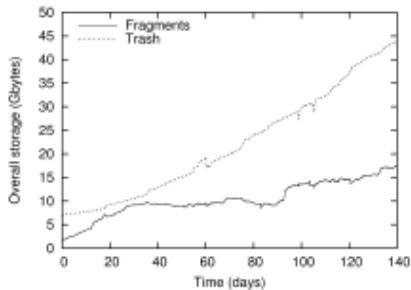


Figure 9. Storage consumed by Glacier fragments and trash.

- Effect of durability requirement on storage overhead is high

Network Overhead

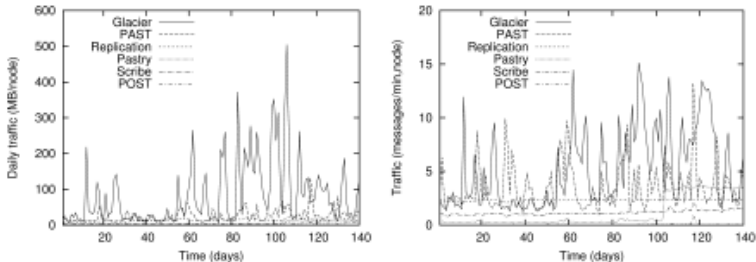


Figure 11. Average traffic per node and day (left) and average number of messages per node and minute (right).

- Most traffic is dedicated to maintenance.
- Msg/min comparable to PAST, but generally larger messages due to replication requirements.

Outline

- 1 Introduction
 - Motivation and Goals
 - Related Work
- 2 System Design
 - Assumptions and Failure Model
 - Design Details
 - Durability
 - Security
- 3 Experiments
 - ePOST
 - **Simulation**

Diurnal Behavior and Scalability

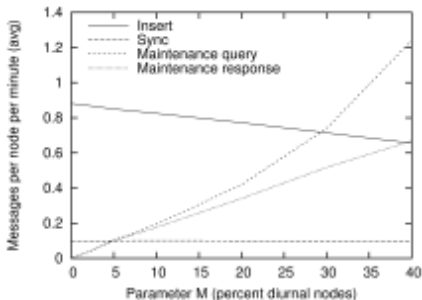


Figure 14. Impact of diurnal short-term churn on message overhead.

- As expected, maintenance cost increases with parameter M .
- Scalable with ring size even with high (constant) churn rate.

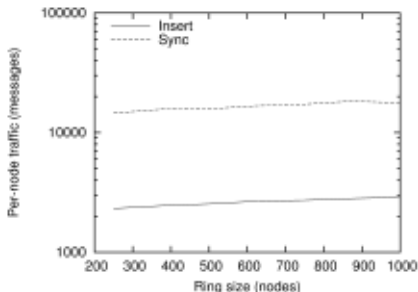


Figure 17. Message overhead for different overlay sizes and a constant per-node storage load.

Summary

- Durable storage despite massive Byzantine failure
- Durable storage without introspection
- Several techniques available to lower overhead
- Appropriate for many (but not all) environments