Total Recall: System Support for Automated Availability Management

Ranjita Bhagwan, Kiran Tati, Yuchung Cheng, Stefan Savage, Geoffrey M. Voelker
University of California, San Diego

UCSD

Availability Management

Automated Availability Management

Goal: Highly available data storage in large-scale distributed systems in which
* Hosts are transiently inaccessible
* Individual host failures are common

Current peer-to-peer systems are prime examples
* Highly dynamic, challenging environment
* hosts join and leave frequently in short-term
* Hosts leave permanently over long-term
* Workload varies in terms of popularity, access patterns, file size

These systems require ***automated*** availability management.
* Availability prediction
* Redundancy management to tolerate transient host disconnectivity.
* Dynamic Repair to tolerate long-term host failures.

We are exploring the challenges of automated availability management in the design, implementation and evaluation of a read/write peer-to-peer file system called **Total Recall**.

Availability Prediction

* Empirically predict availability based on based on measurements.
* Make predictions based on aggregates rather than for individual hosts.
* Short-term availability changes due to transient host failures.
* Long-term availability changes due to long-term host departures/failures.

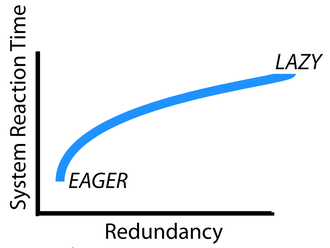
Redundancy Mechanism

* Replication, Reed-Solomon codes, Tornado codes, Online codes.
* Choose redundancy mechanism based on storage, bandwidth and performance tradeoffs.
* Use availability prediction to calculate required redundancy level.

Dynamic Repair

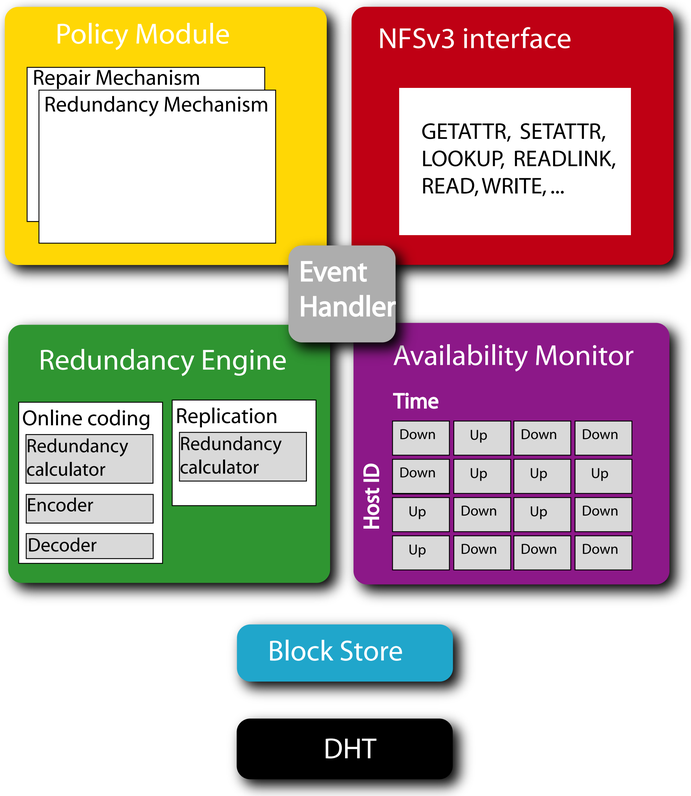
Eager repair: System repairs data redundancy immediately in reaction to host departures.

Lazy repair: System uses additional redundancy to mask transient host departures and defer the costs of repair.



For Total Recall, eager repair => replication, lazy repair => coding.

System Design



Storage System Operations - create, read, write and repair.

* Inodes are eagerly repaired.
* Data may be eagerly or lazily repaired.

Current Prototype

* Runs on PlanetLab.
* Exports the NFSv3 interface.
* Builds on the SFS toolkit and MIT's Chord implementation.
* Uses replication and Online codes as redundancy mechanisms.

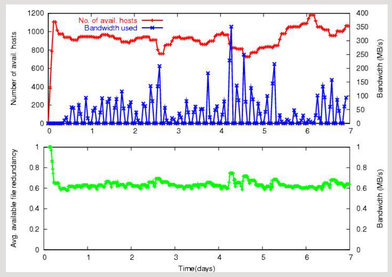
System Evaluation

Simulation

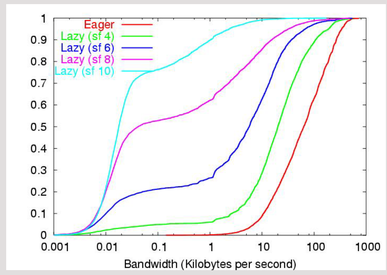
* Simulated 5500 hosts from traces obtained from Overnet P2P file-sharing system.
* Simulated 5500 files. File size distribution obtained from Saroiu et al.'s description of KaZaA workload.

Repair behavior of Total Recall over time

* System bandwidth varies with host availability. Host departures trigger high-bandwidth data repairs, Host arrivals trigger lower-bandwidth metadata repairs.
* Available file redundancy = amount of redundant data the system has available to it to reconstruct the file.
* Avg. file redundancy achieves stable value even as host availability varies substantially.



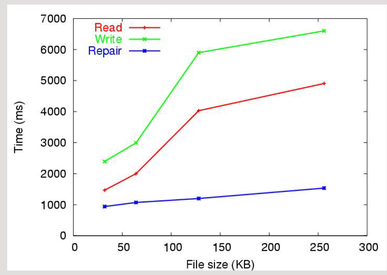
Host bandwidth usage for different repair policies as a CDF



* Total Recall trades off storage overhead and repair bandwidth.
* Eager repair requires the least storage, but most repair bandwidth. Ideal for small metadata.
* Lazy repair smoothly trades off storage (coding stretch factor) with repair bandwidth. Total Recall adjusts degree of redundancy to host availability characteristics.

Prototype Evaluation

* Ran Total Recall prototype on 16 PlanetLab nodes from USA and Europe.
* Used "cp" command through NFS interface to measure file read and write time.
* Measured file repair time.
* All numbers reported for one-file read/write/repair.



Writes are most time-consuming since they comprise of the following operations: inode read, data read, data write and inode write
Reads are less time-consuming, since they comprise the following operations: inode read, data read, and inode write.
Repairs are the fastest since they do not involve remote inode read/write operations.