Background
- Erasure codes provide redundancy without overhead of replication.
  - Divide an object into m fragments.
  - Recreate them into n fragments.
  - A rate r = m/n code increases storage cost by a factor of 1/r.
  - Key property that original object can be reconstructed from any m fragments.
  - E.g. using an r = 1/2 code, divide a block into m = 16 fragments, and encode the original m fragments into n = 32 fragments.
  - Increase storage cost by a factor of 2.
- Example implementations:
  - Reed-Solomon Codes.
  - Turbo Codes.
  - Interleaved Reed-Solomon.

Assumptions
- An archive is implemented on a collection of independently failing disks.
- Failed disks immediately replaced by new, blank ones.
- Each archival fragment for a given block is placed on a unique, randomly selected disk.
- A repair epoch:
  - Time period between a global sweep, where a repair process scans the system, attempting to restore redundancy.

Can This Be Real?
- Three requirements must be met:
  - Failure Independence.
  - Efficient Repair.
  - Data Integrity.

Efficient Repair
- Use cryptographically secure hash algorithm to detect corrupted fragments.
- Verification Tree:
  - n is the number of fragments.
  - Total of n log(n) + 1 hashes with each fragment.
  - Top hash is a block GUID (B-GUID).

Data Integrity
- Erasure codes require precise identification of failed/corrupted fragments.

Case for Erasure Codes
- Exploits the statistical stability of a large number of components.

Availibility
- Grows super-linearly with number of machines.
- Grows super-linearly with decreasing repair times.
- E.g. MTTF = 10^9 years for a particular block.

Conclusion
- The OceanStore archive combines several techniques to satisfy the goals of a global-scale archival system:
  - Erasure codes provide durability and availability.
  - Verification trees provide verifiability.
  - Intra-global failure analysis, automatic repair, and location independent routing promote maintainability.
  - The serializer provides atomicity.
  - End-to-end encryption (not discussed in this poster) provides privacy.

The OceanStore archive combines several techniques to satisfy the goals of a global-scale archival system:
- Erasure codes provide durability and availability.
- Verification trees provide verifiability.
- Intra-global failure analysis, automatic repair, and location independent routing promote maintainability.
- The serializer provides atomicity.
- End-to-end encryption (not discussed in this poster) provides privacy.

Mechanisms are implemented in current prototype.
- First prototype, code name Puddle.
  - Implemented archival model, NFS front-end and archival interface, and ran Andrew Benchmark with 400 client, serializers, and storage servers in network.
- Second prototype, code name Pond.
  - Implemented a hybrid of a Spanner/Elasticized architecture.
  - Implementing Dissemination and repair algorithms.

Enabling Technology: Tapestry
- Tapestry is a location-independent routing infrastructure.
  - Fragments and serializers are both named by opaque bit-strings.
  - Tapestry can perform location-independent routing of messages directly to objects using only GUIDs.
  - Tapestry is an IP overlay network that uses a distributed, fault-tolerant architecture to track the location of every object in the network.
  - Tapestry has two components: a routing mesh and a distributed directory service.

Tapestry Operations
- Publish:
  - send a message toward the root.
  - leaving back-pointers at each hop.
- Lookup:
  - Clients and serializers locate fragments by sending a message toward a root.
  - until they encounter enough pointers.
  - E.g. Client 0325 can locate two fragments of her after only two hops: 0325 => 84F8 => 9098.
- Each GUID is associated with one particular Root node.

Interface:
- Generate new archive interface:
  - create(name, identity, keys) => A-GUID.
- Query Interface:
  - query(A-GUID, Specifier) => V-GUID.
  - parse(A-GUID, data) => V-GUID.
- Read Interface:
  - read(V-GUID, offset, length) => data.
- Write Interface:
  - write(A-GUID, data) => V-GUID.
  - append(A-GUID, data) => V-GUID.
  - replace(V-GUID, offset, data, allowbr) => V-GUID or null.
- allowbr denotes whether operation allowed to generate branch.

Global-Scale Archival Goals
- Durability:
  - Data is stored for centuries or longer.
- Verifiability:
  - Data is not subject to substitution attacks.
- Availability:
  - Data is accessible most of the time.
  - Where most is defined as 9's of availability.
- Maintainability:
  - System recovers from server and network failures.
  - Efficiently incorporates new resources.
- Atomicity:
  - Updates are applied atomically.
- Privacy:
  - Information is only visible to those who have access rights.
- Performance:
  - Response time is bounded.

Archival Model
- Archive Data Structures:
  - Archive is a linearly ordered sequence of versions.
  - Each version is a read-only sequence of bytes.
  - E.g. an archive might be a file, a directory, or a database record.
- Naming:
  - Globally Unique Identifier (GUID).
  - Archives are uniquely specified by archive GUIDs (A-GUIDs).
  - Within an archive, each version is specified by a version GUID (V-GUID).
  - Versions are inalterable and provide for time-travel.
- Operations:
  - Update Operations:
    - Add versions to the end of the version sequence of a given archive.
  - Read Operations:
    - Read data from a specific version.
- Serializer provides consistency:
  - Entity in network that provides atomicity.
  - Provides an A-GUID to V-GUID mapping.
  - Creates a serial order over simultaneously submitted updates.
  - Verifies that the client has update privileges.
  - Atomically applies update to the archive and generates a new V-GUID.
- Sends fragments from an update to storage servers.

Local.
- Durability enhancement techniques such as RAID.
- Servers pro-actively copy data to new disk.
- Servers periodically verify the integrity of local data.

Distributed.
- Exploit Tapestry’s distributed information and locality properties.

Global.
- Has as effective as distributed mechanisms.

Failure Independence: Effective Dissemination
- Model Builder:
  - Takes input from various sources.
  - Builds a model of failure correlation.
- Set Creator:
  - Queries random nodes for properties.
  - Uses the model to compute Dissemination Ssets.
  - Sets of storage servers that fail with low correlation.
- Disseminator:
  - Sends one fragment to each storage server in a set.

Efficient Repair
- Erasure Codes require precise identification of failed/corrupted fragments.

Efficiency
- Compute the failure function for any V-GUID.
- Each GUID is associated with one particular Root node.

Data Integrity
- Use cryptographically secure hash algorithm to detect corrupted fragments.
- Verification Tree:
  - n is the number of fragments.
  - Total of n log(n) + 1 hashes with each fragment.
- Top hash is a block GUID (B-GUID).
- Fragments and blocks are self-verifying.

Tapestry is a location-independent routing infrastructure.
- Fragments and serializers are both named by opaque bit-strings.
- Tapestry can perform location-independent routing of messages directly to objects using only GUIDs.
- Tapestry is an IP overlay network that uses a distributed, fault-tolerant architecture to track the location of every object in the network.
- Tapestry has two components: a routing mesh and a distributed directory service.
- Routing in Tapestry:
  - Nodes are connected to other nodes via neighbor links.
  - Any node can route to any other by resolving one digit at a time.
  - E.g. 0325 => 23B8 => 9098 => 7598 => 4598.
- Each GUID is associated with one particular Root node.