

Atomicity, Serialization and Recovery in the Island-Based File System

Minwen Ji

Princeton University
mji@cs.princeton.edu

1. Introduction

We discuss the consistency protocol in the *island-based file system* [6], a cluster file system designed to provide highly available and scalable data storage to Internet applications. The goal of availability is to maximize the percentage of client requests that succeed despite the failure of one or more *islands*, the building blocks of the file system. A small portion of the directory system is replicated across islands to allow independent accesses to data in each island. As in any other systems where data replication and updates to replicated data are present, the island-based file system faces the challenge of keeping its replicated directory system consistent across islands.

Our goal in maintaining the consistency of the island-based file system is to minimize the efforts for porting applications from single, tightly-coupled and/or small-scale systems to large cluster environments. In particular, we want to eliminate as many hazards as possible that a cluster environment might introduce. Meanwhile, we do not want the consistency protocol to have an intolerable impact on the performance and scalability otherwise achievable in the island-based file system.

The island-based design offers an opportunity for strong consistency without sacrificing performance in common cases. Since it strives to reduce shared state across islands for the purpose of failure isolation, the cost for maintaining consistency of shared state can potentially be reduced as well.

2. Replication model

In the island-based file system, certain directory attributes, but not the directory contents (the lists of names and addresses of sub directories and files) or files, are replicated across islands. The degree of replication varies by directories based on their usage, and changes dynamically as the usage changes. For each replicated *object*, i.e. a set of attributes of a directory, a particular island is chosen as the *coordinator* of this object. Objects in an island are readable by all operations performed in this island. However, updates to a replicated object, called *cross-island operations*, must be originated from the coordinator and be propagated to other islands, called *involved* islands, that have a replica of the object.

3. Consistency protocol design

We design a protocol to ensure that all operations in the island-based file system are atomic and serializable in the face of island failures and network partitions. We use a novel combination of logical clock synchronization [1], two-phase commit [2], logging [3] and finite-state-machine-based recovery to serialize the cross-island operations while keeping the synchronization for *one-island* operations, which involve exactly one island, local.

The basic consistency guarantee our protocol offers is the *atomicity* of operations, i.e. clients would never observe the intermediate state of any operation. We use a vector of logical clocks [1] for the atomicity of cross-island operations. Each island has its local logical clock and each cross-island operation coordinated by this island increases the clock by 1, or *generates* a new clock value. Each island or client of the file system maintains a vector of all islands' clocks. Each request to an island piggybacks the clock vector. We maintain the following invariants:

- 1) The local commit of a cross-island operation and the update of the local clock are atomic in each island, which is guaranteed with a local lock in that island.
- 2) A coordinator does not release the new clock value to a client until it has notified all involved islands of the operation, i.e. until the operation is either *outstanding* or *committed* in all involved islands, except in case of network partitions. This is guaranteed with a two-phase commit [2].
- 3) A request cannot be processed in an island if the request carries a clock that is generated by an outstanding operation in that island.

The three invariants above guarantee that an island will never expose the intermediate state of any operation to clients. Invariant 2 ensures that synchronization on a replica for reads does not need communication with the coordinator, if no network partition is present. The handling of network partitions is discussed in detail in our technical report [7].

The higher-level consistency guarantee our protocol offers is the *serialization* of operations, i.e. clients observe the results of all operations in the same order in all islands. While the logical clock algorithm guarantees

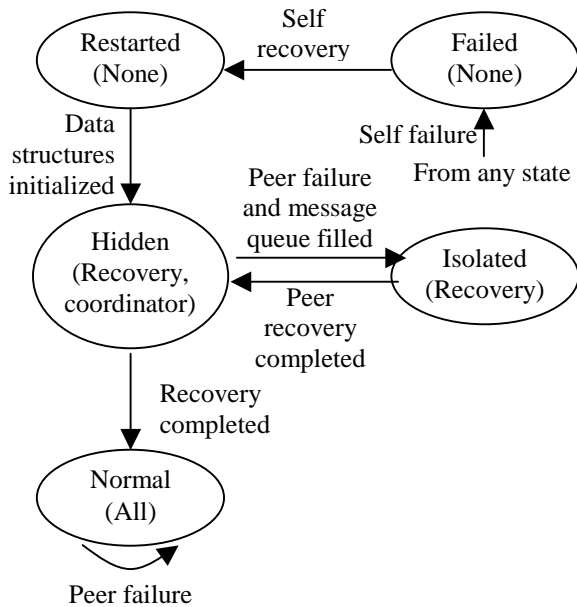


Figure 1. State transitions of an island in response to various failure and recovery events. The types of requests an island receives include *client* requests (from the clients), *coordinator* requests (from the coordinators of cross-island operations), *recovery* requests (from the recovering or reconnecting islands), etc. The types of requests accepted in each state are listed in parenthesis.

serialization in normal cases, island failures and network partitions are handled with additional *write-ahead logging* [3] in surviving islands. The operations in logs will be committed in the ascending order of their clocks during the recovery of an island.

We model a recovering island as a finite state machine, in which each state is distinguished from others by the set of requests that are allowed to be processed in that state, and each state transition is triggered by a failure or recovery event. Figure 1 shows the state diagram.

4. Correctness testing

We use a randomized test engine to test the correctness of the consistency protocol in the face of failures. The test engine is extended from a model checker originally developed in Hewlett-Packard Labs [5]; the model checker is based on the input/output automata (IOA) [4]. We extended the tool so that it checks the implementation of a system, rather than a simulation written in IOA style. The tool helps identifying incorrect parts of a system by injecting randomized sequences of events to the system and analyzing the results. Such events typically could not possibly be experienced in real

workloads or manual tests during a short period of time. We found 14 non-obvious bugs in the protocol during the first two days of testing. The bugs are all at implementation detail level and do not invalidate the overall protocol design.

5. Performance

We have measured the impact of the consistency protocol on the performance and scalability of the system. The measurement of micro benchmarks shows that the protocol adds little overhead to one-island operations, but considerably slows down cross-island operations. The measurement of trace-based operation mixes shows that the overall system scales well with cluster size, e.g. a speedup of 15.7 on 16 islands. Interested readers should refer to our technical report [7] for details.

6. Conclusion

Clusters have been widely used to provide scalable services. An important question for people who are building cluster-based services to understand is how to design software that supports painless porting of applications from single systems to clusters, without weakening the availability and scalability offered by the cluster structure. Our experience suggests that it is possible to distribute data in a cluster file system under such protocols that the system can both achieve good failure isolation and strong consistency, and scale efficiently with the cluster size.

References

- [1] L. Lamport, "Time, Clocks, and the Ordering of Events in a Distributed System", in Communications of the ACM, July 1978.
- [2] J. Gray, "Notes on Database Operating Systems", in Operating Systems: An Advanced Course, 1978.
- [3] R. Hagmann, "Reimplementing the Cedar File System Using Logging and Group Commit", in Proceedings of the 11th ACM Symposium on Operating System Principles, November 1987.
- [4] N. Lynch, and M. Tuttle, "An Introduction to Input/Output Automata", CWI-Quarterly, 2(3), September 1989.
- [5] R. Golding, J. Wilkes, and A. Veitch, private communications, August 1999.
- [6] M. Ji, E. W. Felten, R. Wang, and J. P. Singh, "Archipelago: An Island-Based File System For Highly Available And Scalable Internet Services", in Proceedings of 4th USENIX Windows Systems Symposium, August 2000. Best Student Paper Award.
- [7] M. Ji, and E. W. Felten, "Design and Implementation of the Island-Based File System", Technical Report 610-99, Department of Computer Science, October 1999.