A Reliable and Secure Application Spanning Multiple Administrative Domains

Marc Stiegler

HP Laboratories
HPL-2010-21

Keyword(s):
security, reliability, P2P, multiple administrative domains

Abstract:
A popular technique for avoiding the difficulties of building applications that span multiple administrative domains (MAD) is to create another vendor/application-specific domain, as exemplified by services such as Google Docs and Microsoft Live Mesh. Such a centralized domain adds security vulnerabilities, reliability risks, and scalability costs. Our alternative combines authorization-based access control, a firewall-friendly protocol, and a suitable distributed checkpointing system to build a peer to peer system that needs no central control. Here we discuss the lessons learned in building the Secure Cooperative File Sharing (SCoopFS) application using this approach.

External Posting Date: February 6, 2010 [Fulltext] Approved for External Publication
Internal Posting Date: February 6, 2010 [Fulltext]

© Copyright 2010 Hewlett-Packard Development Company, L.P.

A Reliable and Secure Application Spanning Multiple Administrative Domains

Marc Stiegler
Hewlett-Packard Laboratories
marc.d.stiegler@hp.com

Abstract

A popular technique for avoiding the difficulties of building applications that span multiple administrative domains (MAD) is to create another vendor/application-specific domain, as exemplified by services such as Google Docs and Microsoft Live Mesh. Such a centralized domain adds security vulnerabilities, reliability risks, and scalability costs. Our alternative combines authorization-based access control, a firewall-friendly protocol, and a suitable distributed checkpointing system to build a peer to peer system that needs no central control. Here we discuss the lessons learned in building the Secure Cooperative File Sharing (SCoopFS) application using this approach.

1 Introduction

Distributed applications that span multiple administrative domains (MAD) face several requirements that are either missing or are less severe within a single domain. Such requirements include:

- Ensuring that a consistent state is maintained in the face of failures, even though no single administrative domain can coordinate a global checkpoint of the entire system or orchestrate global recovery. Simplifying, localizing, and bounding checkpointing and rollback is crucial.
- Enabling offline operation, as network partition occurs frequently and uncontrollably.
- Connecting dependably with nodes that may move among domains, such as laptop computers.
- Crossing firewalls protecting different domains.
- Avoiding dependencies upon the different domains' local namespaces, e.g., the user namespaces against which participants authenticate for intradomain resource access.
- Limiting the harm that nodes in one domain can inflict upon nodes in other domains.

Most existing MAD applications employ one of two approaches. The first is to use Federated Identity Management (FIdM) to enable authentication, and to use virtual private networks to reduce the difficulty of operating inside the different firewalls [15]. This approach achieves consistency by minimizing distribution of the computation: the entities work with the authoritative central resource in the application's home domain. A problem with this strategy is that the plan coordination costs for the participant domains' IT departments are often prohibitive. This is true even for variant systems that use roles or attributes for authentication rather that identities: In all cases the IT departments must globally coordinate the meanings of the identities or roles or attributes, which can be a tremendous challenge [4].

An alternative approach, exemplified by Microsoft Live Mesh [3] and Google Docs [19], is to create yet another vendor/application-specific administrative domain and require users to operate within this domain when working with the application. It addresses consistency by maintaining authoritative state in a central single-domain repository. Introducing such a centralized domain brings problems of its own. Notably, this strategy adds an extra central point of failure with new reliability risks, security hazards, privacy concerns, and scalability costs.

We built the Secure Cooperative File Sharing (SCoopFS, pronounced "scoops") application as a MAD peer-to-peer (P2P) system that requires neither a new domain nor an IT-coordinated FIdM solution. SCoopFS's cross-domain file sharing goals are similar to the goals of Live Mesh; it is not a distributed file system akin to CODA [6]. Built on top of the Waterken distributed system platform [7], SCoopFS integrates the following techniques to avoid the problems of both application-specific domains and FIdM integrations:

- A communication-induced checkpointing (CIC) system [9], integrated with a turn-based concurrency model [14, 17], facilitates recovery from node and network failures while allowing each node to control its own checkpointing operations. Earlier work has suggested that recovery with CIC systems may be cumbersome in practice [9]; one contribution of our work is to demonstrate that a judicious synthesis of CIC with turn-based concurrency eliminates this difficulty.
- A redirectory allows mobile nodes that visit multiple domains to register their latest location when they go online, enabling other nodes to find them, ensuring re-

liable operation even as components of a distributed application migrate.
- A forwarder enables firewall crossing, ensuring reliable operation even though new firewalls may be erected, or migration may take components into new firewall-enclosed domains.
- An authorization-based access control system (ZBAC) [12] bypasses the difficulties of authenticating at time of access. Authentication is done once at time of grant. Requested services need not care who is making requests, they need only know that requestors are authorized. Such authorizations flow smoothly across domains despite incompatibilities among local authentication systems, because the service does not invoke any local authentication of a remote client.

The remainder of this paper is structured as follows: Section 2 describes the facilities of the Waterken platform for reliably recovering from failures, controlling access, and crossing domains. Section 3 describes the SCoopFS application. Section 4 describes the types of failures that SCoopFS tolerates. Section 5 discusses the successes and failures in meeting the goals when running SCoopFS in a small pilot program. Section 6 reviews related work.

2 The Waterken Platform

The Waterken platform was designed to enable quick development of reliable secure distributed applications. Figure 1 illustrates the basic elements of a Waterken platform and its environment in the context of the SCoopFS application. A *node* is a shared-memory assembly of hardware resources controlled by a single operating system. It may support multiple Java virtual machines (JVMs), each of which may run a single Waterken platform, which may run multiple concurrent *actors*. Each actor is an assembly of objects sharing a single thread of control and a single incoming event queue; only one event is processed by an actor at a time. Multiple objects within the actor may export their interfaces to the outside world. Objects within an actor may interact with each other using *immediate calls* (ordinary method invocations), and block while waiting for the response. All object references across actors must be performed with nonblocking *eventual sends*, using promises [17] to receive responses from their sent messages. An *application* is a collection of actors that may span nodes, that have a network of references among themselves, and that work toward a common goal.

Reliability, including easy recovery in the face of network and node failures, is a key goal of the Waterken design. The Waterken platform uses a CIC checkpointing mechanism, which allows actors to checkpoint independently, an important feature in a MAD environment where no one can coordinate a global checkpoint. CIC systems must overcome a number of obstacles.

CIC systems may suffer from the "domino effect": while recovering the checkpoint for one process, the system may determine that it must roll back other processes to earlier checkpoints, which may require yet more rollbacks [9]. To avoid such cascading rollbacks, Waterken uses the known technique of checkpointing an actor before every message-sending event [5]. Because the concurrency model is turn-based, the messages sent during a turn can be accumulated until the turn ends, at which point a checkpoint is taken and all the messages are sent in a single message-sending event. Bundling the sending of messages in this way reduces the number of checkpoints required. Furthermore, though an actor can be as large as an entire JVM, experience to date suggests most actors are small, so most checkpoints are also small.

Like other CIC systems, Waterken piggybacks checkpointing information on the application messages; unlike other CIC systems, this information is purely local, informing the recipient's platform whether to expect to queue new messages or modify mutable state as a result of processing the message.

Message sending is not the only trigger for checkpointing. Mutable actor state modification also forces an end-of-turn checkpoint. All checkpoints include the modified state, the list of messages to send, the value to be returned to the sender of the message that initiated the turn, and the ID of the incoming message which initiated the turn. There are no useless checkpoints, and every checkpoint advances the recovery line [9]. Messages are idempotent, so if an availability failure occurs during processing of a message, the sender can retry knowing that the message will be processed only once: if the recipient did not receive or process the message the first time, it proceeds upon receipt of the retry, whereas if the message was processed the first time, the recipient's Waterken platform automatically returns the previously-computed result, and the actor never sees the duplicate. Neither is the sending actor aware of the retries: Waterken automatically retries if necessary, so sent messages eventually arrive and results are eventually returned; the platform fulfills the promise (i.e., supplies the response) to the actor when that response eventually arrives. Messages between pairs of actors are FIFO ordered. Messages from multiple senders to a single recipient may be interleaved arbitrarily (subject to the pairwise-FIFO constraint).

Figure 2 summarizes how the Waterken platform drives an actor through a turn. There are several consequences of this algorithm. Every message is eventually processed exactly once unless the sender or recipient is taken offline forever. There is always a consistent version of the entire distributed system on stable store, including pending messages. No actor need ever roll back more than one check-

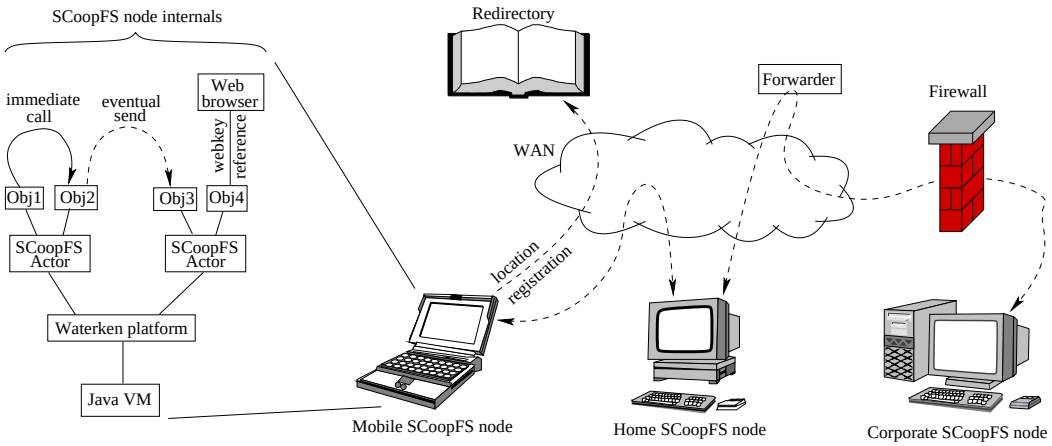

Figure 1: Waterken and SCoopFS architecture.

point. No rollback of one actor requires rollback of another actor. The net result is that recovery from a failstop failure becomes straightforward: simply restart the platform that failed. Eventually all pending incoming messages will be retried and processed, responses will be returned, outgoing messages will be sent, and the whole system progresses.

The way a Waterken platform handles incoming and outgoing messages for an actor and implements restarts following a failure prevents inconsistencies from propagating to other actors. Intuitively, there is no way for an actor to tell whether another actor has failed or is merely slow. The result is that all actors pass through a set of states following the recovery that they could have passed through had the failure not occurred. This enables "eventual distributed recovery," requiring only that every actor eventually complete all its turns and every message eventually be delivered.

Waterken uses webkeys [22] for access control to objects. Webkeys are unspoofable, unguessable, self-authorizing references embodied as URLs transmitted using HTTPS protocol with self-signed certificates. Webkeys are somewhat analogous to car keys: the car does not need to check your driver's license before you drive it, it only needs to know that you have the key that starts it.

Reliability is further enhanced by the redirectory integrated with Waterken platforms. The redirectory is a Waterken-based application with which Waterken platforms register their IP addresses when they launch. If a Waterken platform and its actors move to a new IP address or a different domain name, platforms with actors that are clients find the relocated platform in its new location by querying the redirectory. The actors and applications neither know nor care that a part of the system has moved: nothing breaks. Since webkey references are unspoofable, buggy or malicious redirectories cannot steal any authorities, nor cause messages to be sent to the wrong recipient, despite migration.

For the SCoopFS project we built an additional application for the Waterken infrastructure, a rudimentary firewall-penetrating forwarder. The forwarder extends the reliability of distributed Waterken applications by enabling platforms to operate together even though new firewalls might be erected, even though nodes and platforms might migrate across firewalls. The webkey strategy of using HTTPS protocol facilitated the building of the forwarder, since few firewalls block HTTPS traffic.

3 SCoopFS

SCoopFS is a P2P file sharing system designed to enable easy, intuitive collaborative editing in a MAD environment [13]. It is designed to impose neither IT costs for the administrators of participating domains, nor privacy concerns due to a central administrator, nor hardware provisioning costs for a central server farm. It uses an email metaphor (but not SMTP email) for file sharing: the owner of a document brings up his SCoopFS mailbox in his browser, specifies a recipient and a file attachment, and sends a scoopfs-mail. The recipient saves the attachment somewhere on stable store; when the attachment is sent with read/write authorization (embodied as a webkey reference to an update channel for the original file), SCoopFS sets up synchronizers such that, any time either user edits the file, the other user's copy of the file is updated. Offline editing operations result in queued updates that are delivered upon reaquisition of network access. The SCoopFS mailboxes detect simultaneous edits resulting in conflicts, and assist the users to resolve them. There is no notion of locking.

SCoopFS fully supports "rich sharing," integrating the abilities to dynamically cross domains, chain attenuated delegations, recompose authorities from different domains, and hold entities accountable for misuse [21]. Rich sharing enabled users in the pilot program to devise unantic-

- *Pick the next message sent eventually to an object in this actor from the actor's event queue*
- *If this message was already processed,*
    - *return previously computed response*
- *Else,*
    - *Deliver the message to the object*
    - *While the actor processes the incoming message,*
        - *If the actor makes an eventual send of a new message,*
            - *put that message in the list of messages to send*
    - *Upon return of a response from the actor*
        - *Take a new checkpoint, including:*
            - *response to return to the message sender*
            - *ID of the received message for which the response has been computed*
            - *modifications to actor state*
            - *list of new messages to send*
        - *Delete the old checkpoint*
        - *Return the response to the sender*
        - *Send, and resend as necessary, the new messages on platform threads concurrently with picking the next message*

Figure 2: Waterken main event loop.

ipated uses. One relevant application was a patch update system. By using attenuated chaining, the developers of program updates were able to push patches to one user in each user-community, whose machine would in turn automatically push the patch to the other members of that community and other associated communities. No specially provisioned central site with high bandwidth was required. This type of application is not possible for conventional file sharing systems like Live Mesh, since they do not support chained delegation of attenuated authorities, i.e., the recipient of a read-only authority cannot derive and re-grant a separately revocable read-only authority to another party: Sharing requires access control list editing authority, which conveys de facto full authority. Indeed, this inability to enable attenuated delegation is a key reason why most file sharing is done, not with file sharing tools, but rather with email [11].

For deployment, a Waterken platform is installed onto every machine in a SCoopFS file sharing system. The Waterken platform is then launched, and a set of actors that comprise a SCoopFS mailbox is created, along with a webkey reference to the user-facing mailbox actor. The user interacts with the mailbox using the mailbox webkey from within his browser (typically the webkey's domain is hardwired to localhost on the machine, so the browser gets a direct connection to the mailbox). If two SCoopFS users who do not have scoopfsmail addresses for each other in their address books decide to share, one of them gets a new message-channel webkey from his SCoopFS system, and the other puts that webkey into his SCoopFS system. Each user then appears in the other's address book. Webkeys for update channels on a specific file are transferred in the scoopfsmail message that carries the file as an attachment. Checkpoints are taken automatically by the platform of the relevant actors at the critical junctures during the exchange of the webkeys and the updates, ensuring a reliable exchange of webkeys and data.

4 Failure Model

SCoopFS is designed to be robust in the face of the following failures:

- Transient network failures. Message insertions, deletions, alterations, and replays can only degrade performance.
- Failstop failures of the node hardware, operating systems, and Java virtual machines. Transient failstop failures of these elements on a node can be fixed by restarting the platform. If the failure of one of these elements is deterministic, the problem can be solved by migrating the application state to a node with replacement hardware/alternate OS/upgraded JVM for the broken element. After such a migration, other nodes find the migrated platform via the redirectory.
- An arbitrary number of failstop failures of arbitrary nodes at arbitrary times. Recovery is done by relaunching failed platforms in any order.

SCoopFS and Waterken both assume correct operation of the stable stores for all the nodes. If a node's stable store fails, those application activities requiring objects on the failed node will cease, but the rest of the system will continue to progress. A deterministic failstop failure of a Waterken platform will produce the same level of degradation of the system.

Using the BAR categories of node behavior [2], at least one redirectory and at least one forwarder must be altruistic. Byzantine redirectories and forwarders can only reduce system performance. SCoopFS mailboxes are assumed to be rational. Byzantine mailboxes can do harm only up to the limit of their webkey-derived authorities. Under no circumstance can a Byzantine mailbox corrupt the checkpoint of another mailbox, for example, but it can corrupt shared files for which it has been granted a webkey reference for an update channel (i.e., if it has been granted edit authorization on

a shared file). Actions such as updates and message sends are unspoofably bound to petnames [20]. Thus abusers of authorizations may be held accountable, for example by revoking their authorizations using the ZBAC-oriented caretaker pattern [18].

5 Experiences

SCoopFS was deployed for nine months in an end user pilot program with a dozen users [13]. Users routinely carried SCoopFS-enabled laptops between administrative domains and across firewalls, shutting the machines down with arbitrary abruptness, leaving them off for arbitrary periods. Despite this, no loss, damage, or duplication of messages, updates, or shared files was ever observed. No recovery ever required rollback of more than one platform, achieved by simple relaunching.

Any time a SCoopFS node could reach the Internet, file sharing and message sending/receiving was operational. No firewalls were encountered that impeded the transmission of webkey-based messages. No IT department assistance was required to enable SCoopFS operation across any of the domains.

In one experiment, file updates were passed via the forwarder between two nodes that were shut down and restarted in sequence so that they were never both alive at the same time. Eventually the entire multistep activity completed with no errors.

A reliability risk in many concurrency models is deadlock. The promise pipelining concurrency model is deadlock free, though a related liveness issue, *datalock*, is possible [16]. Datalock involves cycles of unresolved promises. Although the actors associated with such promises continue to execute—because actors and all interactions among them are nonblocking—the code associated with the promises will never execute. A contribution of our work is experiential support for the prediction that datalocks should be rare in practice [16]: No datalock was ever observed at any stage in development or deployment of SCoopFS.

The greatest source of systemic unavailability was plan coordination with the IT department. IT periodically required the forwarder's node to be patched with security upgrades, requiring shutdown and relaunch of the forwarder. Once IT shut down the forwarder's network for a week. In all cases, pending messages and updates that had been awaiting service were delivered successfully after the forwarder came back online. The unavailability of the forwarder did not prevent creation and queuing of messages and updates on individual SCoopFS nodes for eventual delivery. Hence forwarder unavailability generally went unnoticed by users.

Though the architecture of SCoopFS eliminated all central points of failure, the pilot implementation did not. Only one forwarder and one redirectory were supported in the pilot. In a large-scale deployment, we believe the forwarder would be the first overutilized resource, and a network of forwarders would be required.

Another deficiency of the pilot implementation was that the forwarder transiently held data and authorities in the clear, making it a privacy and security vulnerability. Again, the architecture supported elimination of this weakness.

We observed one performance problem late in the pilot program. Waterken was checkpointing each actor with many small files that accumulated. On the NTFS file system, performance degraded with 100K files in a single folder; 400K files made SCoopFS unusable. On the other hand, the forwarder accumulated 350K files on ReiserFS with no noticeable degradation. Analysis enabled modification of the Waterken checkpointing algorithm and eliminated this problem.

Aside from this one bit of analysis and tuning done in response to lessons from the SCoopFS application, no other performance work has been done, though many candidate optimizations for improving performance of the current system are known; this is now an area of active research.

Additional performance tuning was unneeded for SCoopFS: no other noticeable performance problem occurred at the user level. Part of the reason SCoopFS operated so effectively was its P2P architecture: as more users were added to the system, more user compute resources were implicitly added as well.

Until further performance tuning has been performed, the Waterken platform may be ill-suited for high performance distributed computing applications. However, SCoopFS did demonstrate that it meets some of the requirements for programmer-transparent checkpointing and easy rollback that have been identified for future peta-scale systems [10].

From a dependability perspective, our experience running the pilot program was consistent with our expectation.

6 Related Work

Like Waterken, the Sinfonia platform was designed to ease the process of building scalable reliable distributed systems [1]. A key goal was to allow proving the dependability properties of the platform once, and thereby avoid having to prove the dependability properties repeatedly for every distributed application. Sinfonia is focused primarily on rapid development of systems within a single administrative and trust domain. The CatchAndRetry proposal [14] for handling distributed system failures, designed for turn-based concurrency such as that used by Waterken, is partly incorporated in the platform: availability errors are handled automatically by retry, the application programmer neither knows nor cares that such failures have occurred (the pro-

grammer can explicitly timeout if prompt action is required; such timeouts were unnecessary in the SCoopFS system because the default retry system was always appropriate). Tongo [8], a framework for developing and executing SOA applications for mobile devices, uses a centralized application server/domain and is focused primarily on enabling implementation modifications at runtime. The E programming language [16] uses the same promise pipelining model of concurrency as the Waterken platform, but does not include CIC checkpointing, or the forwarder, or the redirectory, that give the Waterken platform many of its reliability characteristics.

7 Conclusion

When communication induced checkpointing is combined with a suitable turn-based concurrency model and a redirectory system, restart of failed services in a MAD environment ceases to be cumbersome in practice. Authorization-based access control with webkeys supplies a form of Web-based computation that easily and securely conveys attenuatable authorities across multiple administrative domains. The performance penalties for CIC systems may be ameliorated, or rendered unimportant, for peer to peer applications that scale out the resources as they scale up the utilization. Furthermore, as faster stable storage technologies fall in cost, these types of turn-based CIC systems will receive comparatively greater speedups, and will deserve more consideration for a wider variety of distributed applications.

References

[1] M. K. Aguilera, A. Merchant, M. Shah, A. Veitch, and C. Karamanolis. Sinfonia: A new paradigm for building scalable distributed systems. In *SOSP*, 2007.

[2] A. S. Aiyer, L. Alvisi, A. Clement, M. Dahlin, J.-P. Martin, and C. Porth. BAR fault tolerance for cooperative services. In *SOSP*, 2005.

[3] M. Ashley. First impressions: Live Mesh. *PCWorld*, Apr. 2008.

[4] R. Bachert. Moving from one theater to the next using an enterprise directory. In *LandWarNet Conf.*, Nov. 2007.

[5] J. Bartlett. A non stop kernel. In *SOSP*, 1981.

[6] P. Braam. The Coda distributed file system. *Linux Journal*, 50, 1998.

[7] T. Close. Waterken server. `http://waterken.sourceforge.net/`.

[8] D. Correal. Tongo: A framework for supporting mobile application architectures. In *DSN Workshop on Architecting Dependable Systems (WADS)*, 2009.

[9] E. M. Elnozahy, L. Alvisi, Y. min Wang, and D. B. Johnson. A survey of rollback-recovery protocols in message-passing systems. *ACM Computing Surveys*, September 2002.

[10] E. N. Elnozahy and J. S. Plank. Checkpointing for peta-scale systems: A look into the future of practical rollback-recovery. *IEEE Transactions on Dependable and Secure Computing*, June 2004.

[11] M. L. Johnson, S. M. Bellovin, R. W. Reeder, and S. E. Schechter. Laissez-faire file sharing. In *New Security Paradigms Workshop (NSPW)*, 2009.

[12] A. Karp. Authorization based access control for the services oriented architecture. In *Conf. on Creating, Connecting and Collaborating through Computing (C5)*, 2006.

[13] A. Karp, M. Stiegler, and T. Close. Not 1 click for security. In *Symposium on Usable Privacy and Security (SOUPS)*, 2009. Also HP Labs tech report 2009-53.

[14] E. Kiciman, B. Livshits, and M. Musuvathi. CatchAndRetry: Extending exceptions to handle distributed system failures and recovery. In *PLOS*, 2009.

[15] K. Michael. Session six: Case studies in favour of a federated identity. In *Mgmt & Info Security Forum*, Feb. 2009.

[16] M. S. Miller. *Robust Composition: Towards A Unified Approach to Access Control and Concurrency Control*. PhD thesis, Johns Hopkins, 2006.

[17] M. S. Miller, D. E. Tribble, and J. Shapiro. Concurrency among strangers. *Trustworthy Global Computing*, 2005.

[18] D. D. Redell. *Naming and Protection in Extendible Operating Systems*. PhD thesis, MIT, 1974.

[19] G. Ross. Google Docs. *NotebookReview.com*, Oct. 2009.

[20] M. Stiegler. An introduction to petname systems. *Advances in Financial Cryptography Vol. 2*, 2005.

[21] M. Stiegler. Rich sharing for the Web. Technical Report 169, HP Labs, 2009.

[22] M. Stiegler. Towards fearless distributed computing. Technical Report 258, HP Labs, 2009.