Scaling Security for Big, Parallel File Systems

Andrew Leung, Ethan L. Miller

1 Motivation

The need for peta- and exabyte scale parallel file systems that support high-performance computing (HPC) has been rapidly increasing. These systems have unique demands, different from those of traditional distributed file systems. As a result, securing I/O in big, parallel file systems without significantly impacting performance has proven challenging.

Parallel file systems are commonly composed of three components: clients, metadata servers, and network-attached storage devices, such as disks or object-storage devices. Security is commonly based on a capability model, where capabilities are 'tokens' which represent single block or object I/O authorization. Capabilities are generated by metadata servers, given to clients, and presented to storage devices with I/O requests.

Peta- and exabyte scale systems may have tens of thousands of clients and storage devices. Files are very large, often gigabytes to terabytes, and are commonly striped across thousands of storage devices. HPC workloads are parallel and bursty, meaning I/O often comes from many clients at a time with very short inter-arrival times. This implies files are commonly accessed by thousands of clients within a few seconds. These factors are often worst-case scenarios for many existing solutions.

While there are many security schemes for distributed file systems, few have addressed large-scale, demanding environments. In these environments existing solutions cannot sustain high performance or must weaken security to do so. HPC workloads can burden metadata and data servers with generating and verifying millions of single block or object capabilities. Reliance on shared-key cryptography introduces vulnerabilities when millions of storage devices are susceptible to attack. Also, many common security techniques, such as revocation, become difficult with so many clients and storage devices.

2 Our Approach

We are developing a protocol that is designed to provide secure I/O for petabyte-scale, parallel file systems. Our approach seeks to leverage common HPC characteristics and simplify security management to improve performance and scalability. Our protocol, Maat, employs three main concepts: extended capabilities, automatic revocation, and scalable, secure delegation.

Extended capabilities allow a capability to authorize I/O for any number of clients to any number of files. Grouping policies allow the MDS to decide how to include client and file authorization in a capability. By conferring access for client-file pairs, rather than single client-block or client-object pairs, and aggregating many pairs into a capability Maat is able to greatly reduce the number of capabilities in the system, as well as, exploit capability reuse. Extended capabilities are secured using asymmetric cryptography, allowing anyone in the system to verify a capability's integrity. Though asymmetric cryptography is quite slow, the reduced number of capabilities serves to greatly amortize this cost. Merkle trees are used to identify authorized client and file I/O in a capability, allowing capabilities to be fixed-size and still enforce confinement. Through the use of an update protocol, clients and storage devices can flatten these trees into lists of clients and files. A lookup mechanism allows storage devices to then map files to local blocks or objects.

Automatic revocation allows access rights to be revoked without the need to contact any clients or storage devices. Each capability has a short timeout. Once a capability has timed-out it is no longer valid. Capability lifetimes can be extended using extension token that are distributed by metadata servers and cached at clients and storage devices. A single extension token can proactively prolong the timeouts for a very large number of capabilities by batching many extensions into a single token. This allows clients to reuse valid capabilities with little overhead. As a result, revocation is independent of who holds capabilities or which devices hold file data, allowing Maat to scale to very large systems.

Cooperative computation, common in HPC workloads, is achieved through the use of *scalable, secure delegation*. A single client may open a file on behalf of any number of other clients to reduce the overhead of large, joint computations. This client generates an asymmetric key pair and uses the public key to receive a file handle and capability associated with the key from the metadata servers. The client then distributes the private key, capability, and file handle to any other clients cooperating in the computation which the other clients use to perform I/O. This provides scalable delegation and supports POSIX HPC I/O extensions `openg()` and `openfh()`. Additionally, by associating clients with computational keys at the metadata servers we are able to achieve confinement.

3 Current Status

An earlier paper [1] discusses some of our initial designs. We are implementing Maat in the Ceph petabyte-scale, parallel file system [2]. Ceph is an object file system which achieves much of its scalability by pushing responsibility to intelligent network-attached object-storage devices (OSDs). This allows Maat to leverage OSD intelligence without addressing power and cache size issues common to disks. In the future Maat will address other issues, such as, on-disk security.

References

[1] A. Leung and E. L. Miller. Scalable security for large, high performance storage systems. In *Proceedings of the 2006 ACM Workshop on Storage Security and Survivability*. ACM, Oct. 2006.

[2] S. A. Weil, S. A. Brandt, E. L. Miller, D. D. E. Long, and C. Maltzahn. Ceph: A scalable, high-performance distributed file system. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation (OSDI)*, Seattle, WA, Nov. 2006.