Scalable Security for Petascale, High Performance Storage

Andrew W Leung, Ethan L Miller, Stephanie Jones

Problem:
- Storage is getting bigger
- More high performance Apps
- Data is confidential/sensative
- Current security is not scalable

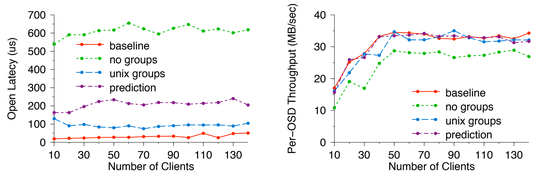
New Challenges:
- Tens of thousands of nodes
- Large files/Highly distributed
- Nodes are more vulnerable
- Demanding I/O patterns

Our Approach:
- Maat: Scalable Security for Petascale, High Performance Storage
- Extended Capabilities
  - A capability authorizes I/O for many clients to many files
  - Limits the number of capabilities even for large systems
- Automatic Revocation
  - Capabilities have short lifetimes, timeout after 5 minutes
  - Capability expiration acts as global capability revocation
  - Allows capability revocation without the need to contact any nodes
- Secure Delegation
  - Clients can distribute file access to others without fear of eavesdropping
  - Allows clients to securely act on behalf of others and delegate access
  - Conforms to proposed POSIX extensions: openg() and openfh()

Implementation:
- Implemented in the Ceph petascale, distributed file system
- Cryptographic algorithms:
  - Public key: ESIGN, Shared key: AES, One-way hash: SHA-1
- Authorization grouping strategies:
  - UNIX groups, Recent Popularity predication, temporal batching

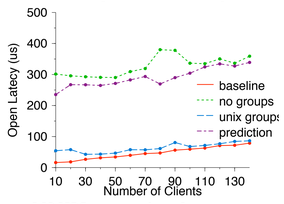
Evaluation:
- Testbed: 18 node Linux cluster with 1 MDS, 10 OSD and 7 client nodes
  - Each client node runs up to 20 client instances concurrently
- On-wire encryption and client data cache disabled
- All experiments use insecure Ceph as a baseline comparison
- We compare extended capabilities to an approach without authorization grouping

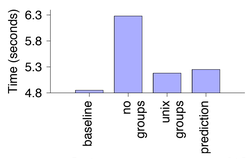
Micro-benchmark:
- Clients write a mix of shared and non-shared 5MB files
- UNIX groups consist of 10 clients



- Extended capabilities with UNIX and prediction authorization grouping incur an open latency and OSD throughput significantly better than no grouping and comparable to baseline Ceph

IOR2 Benchmark:
- A HPC parallel file system benchmark developed at LLNL





- UNIX grouping has an average open latency on par with baseline Ceph. Prediction was unable to make many predictions from IOR2 access patterns and performed worse
- Overall, Maat's securtiy added only a 6-7% overhead to baseline

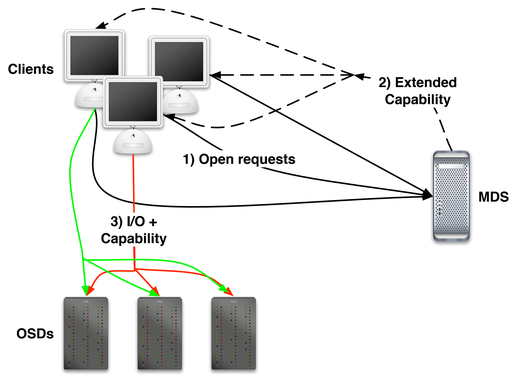
Conclusions:
- Maat addresses the need to security which can scale with petabytes of data and HPC workloads
- Maat provides secure access control, revocation, and access delegation
- Despite strong security Maat adds only a minimal overhead compared to baseline Ceph
- With such a small overhead, there is no longer an excuse to exclude security from petascale, high performance storage

Extended Capabilities

Authorize Many I/Os

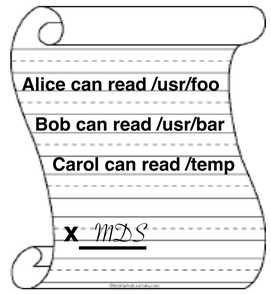
- Many authorizations grouped into a capability
  - Reduces the number of capability generations
  - Reduces the number of capability verifications

Red I/O needs verification, green I/O does not



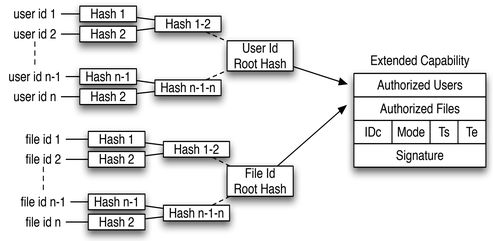
Cryptographically Secure

- Signed by the MDS, can be verified by any OSD
- More computationally expensive
  - Affordable because far fewer capabilities are needed
- Capabilities name all authorized user-file I/Os
- Attacker cannot forge or use stolen capability



Fixed Size

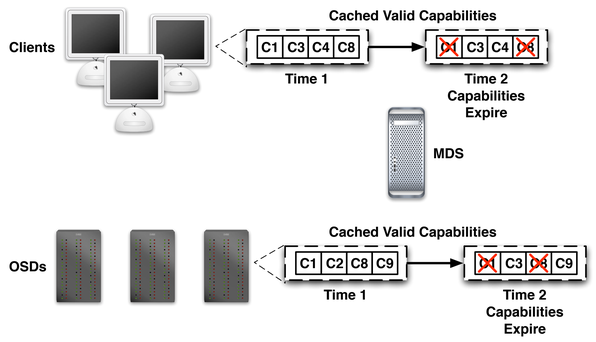
- Small, fixed size, easy to cache, better for network passing
- All users and files named with Merkle trees
  - Root hash of each tree included in the capability
- OSDs need to map root hash to user names and file ids
  - OSD obtain/cache signed list of users/files from clients



Automatic Revocation

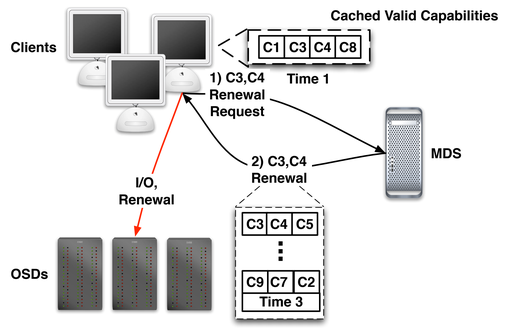
Expiration as Revocation

- Capability expiration acts as global revocation
  - No need to contact any devices, means it is scalable
- Each capability has a short lifetime
  - Limits window of vulnerability when access is revoked



Capability Renewal

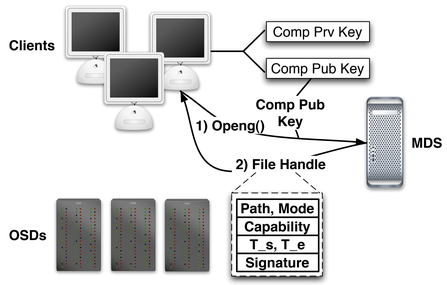
- Only want invalid capabilities to expire
- Issue token to renew all valid capabilities
  - A single token can renew many capabilities
- Shifts cost from revocation to renewal, more scalable



Secure Delegation

Group File Opening

- A client generates a temporary key pair, the Computation Key Pair
- Call openg() and pass the computation public key
- MDS returns file handle and capability
  - Capability authorizes I/O for anyone who has the private key



Delegation of Access

- The file handle, capability, and encrypted private key are passed to others
- Clients use openfh() to turn file handle into file descriptor
- Clients perform I/O with the capability and token proving possession of the private key