Scalable Access Control for Secure Multi-Tenant Filesystems

Giorgos Kappes
Supervisor: Stergios Anastasiadis

Department of Computer Science & Engineering
University of Ioannina
Greece

October 3 2013

Outline

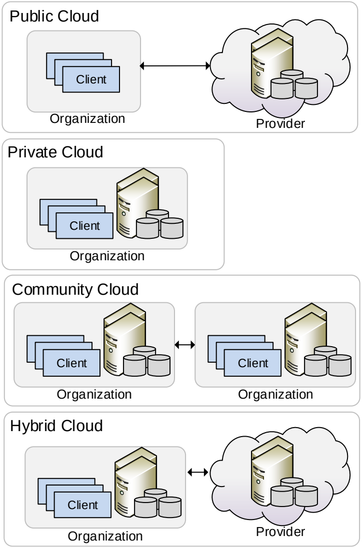
Cloud computing

Access to large amounts of resources

- Resource aggregation
- Maximized effectiveness of shared resources
- Reduced costs for end-users

Cloud deployments

- Public Cloud
- Private Cloud
- Community Cloud
- Hybrid Cloud



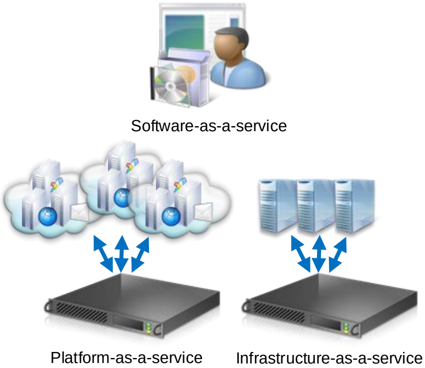
Cloud service models

Cloud services

- Software-as-a-service
- Platform-as-a-service
- Infrastructure-as-a-service

Accessed through:

- Web browser
- Thin client
- Mobile application

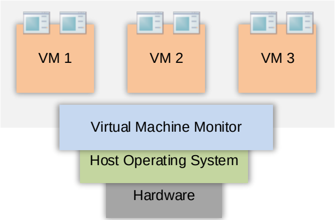


Virtualization

Sharing of computer system resources

Virtual Machine Monitor

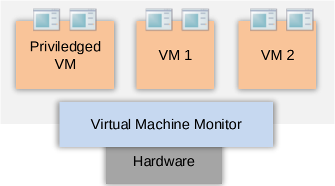
- Software layer placed on top of the hardware layer
- Manages and allocates system resources to VMs
- Provides isolation



Hosted Architecture

Virtual Machine Monitor architectures

- Hosted
- Autonomous



Autonomous Architecture

Storage consolidation in virtualization environments

Concept

- Centralizing & sharing storage resources across applications/users

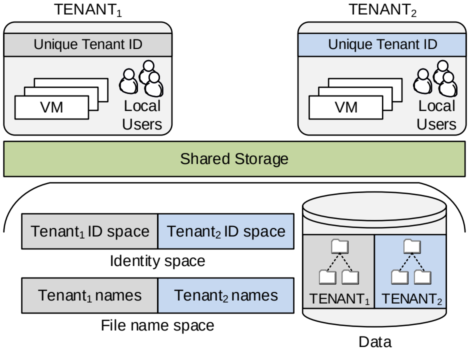
Consolidation at the block level: **Virtual disks**

- Support for isolation, versioning, mobility, heterogeneous clients
- No opportunities to share read-write access
- Complicated sharing and manageability
- Reduced performance due to the large number of storage layers

Consolidation at the filesystem level: **Virtualization-aware filesystems**

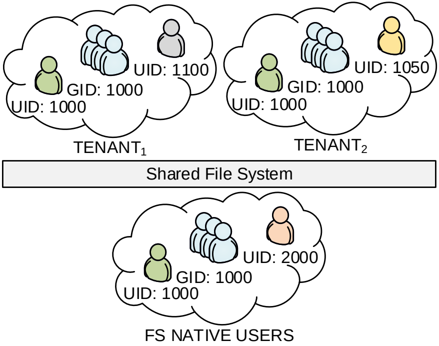
- Data sharing, increased administration flexibility/efficiency
- Ephemeral and highly composable storage
- Improved performance

Our goals



- **Isolation:** Isolate tenants and prevent namespace collisions
- **Sharing:** Flexible intra-tenant/ inter-tenant sharing
- **Efficiency:** Fast data access with native support of multitenancy
- **Compatibility:** Architectural compatibility with existing filesystems
- **Manageability:** Efficient administration of the file system

Multitenancy challenges of shared file-level storage



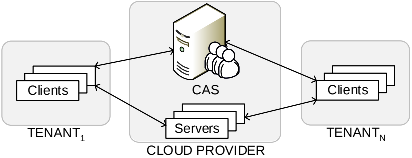
Single shared namespace

- Tenant namespaces not isolated
- Identity collision problem
- Problems with permissions and special files

Main approaches to prevent identity collisions

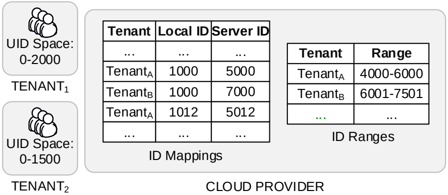
Centralized identification

- ID management: shared service
- Enormous number of users
- Scalability and security problems



Identity mapping

- Local IDs ➞ globally unique IDs
- Limited scalability, complicated file sharing and manageability



Why file sharing is important for cloud environments?

Virtual desktops

- An enterprise stores the desktop filesystems of personal thin clients
- Separate root tree for each tenant, shared folder for collaboration

Software-as-a-service

- A SaaS provider supports different business customers
- Separate application files, but possibly shared system files

Software repository

- Shared software repository forked into shared or private branches

Medical records and scientific data

- Health-care/research data shared between affiliated hospitals/groups

Parallel distributed filesystems

Goals

- Parallelization of file I/O
- Elimination of the potential metadata bottleneck

Separate management of file metadata and data

- Metadata managed by metadata servers (MDSs)
- Data managed by object storage servers (OSDs)

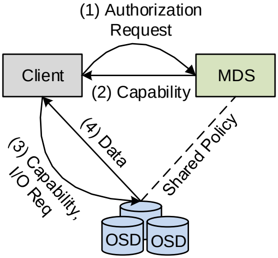
Storage in the form of objects

- Data and metadata split into objects
- Objects are stored on OSDs

Filesystem client

- Full filesystem abstraction to users

Access control on parallel distributed filesystems



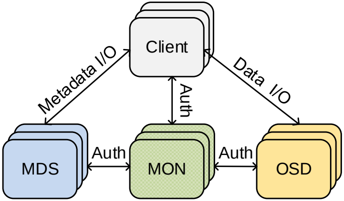
Access control decisions happen at the MDS

- OSDs: no knowledge of access control info
- MDS: authorizes requests and provides clients with capabilities
- Client: presents the capability to the OSD

A case study: Ceph

A distributed object-based filesystem

- Clients: provide access to the FS
- MDSs: manage the FS namespace
- OSDs: store data and metadata
- MONs: manage the cluster map



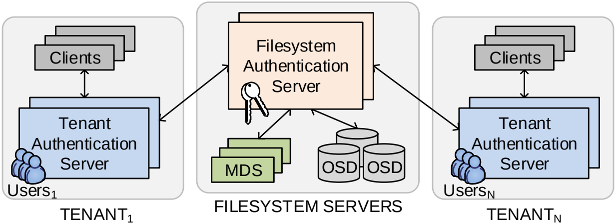
Metadata management

- Folder: stored as a single object, or as a collection of fragments
- The MDS caches recently-updated metadata

Data and metadata storage

- Objects stored as files: identifier, binary data, object metadata
- Objects mapped to PGs and PGs to OSDs with CRUSH

Architectural overview



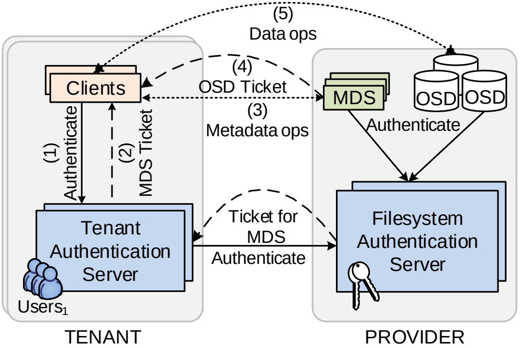
Tenant Authentication Server (TAS) certifies local clients/principals

- Tenants/clients/principals publicly identified by their public key
- Tenants can use their own Identification mechanism internally

Filesystem Authentication Server (FAS) certifies TASs/MDSs/OSDs

- Manages the operation of the whole system

Authentication



Definition

- Verification of an entity's identity

Idea

- Principals: connect to a client and authenticated by a TAS
- From the TAS: principals retrieve ticket for the MDSs
- From the MDS: principals retrieve ticket for the OSDs

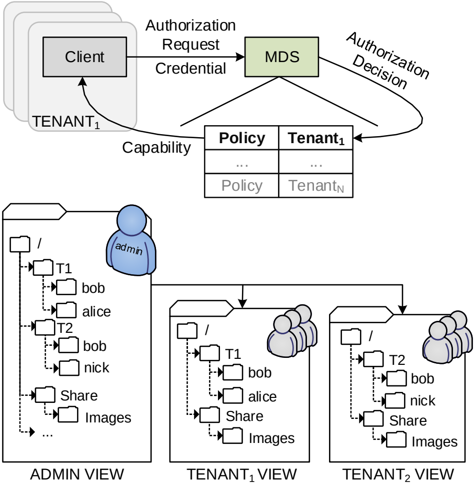
Authorization

Definition

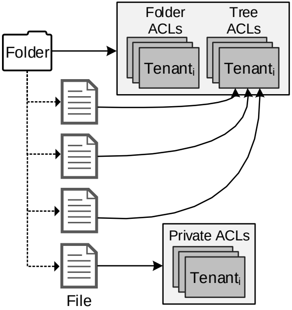
- Determining principal's rights

Idea

- Separate ACLs for each tenant and the provider
- Files: private/shared across principals of 1 or more tenants
- Namespace filtering: Selective access to metadata
- Tenant view and Admin view



Optimizations



ACL Sharing

- Folders: 2 ACLs per tenant. *folder ACL* + *tree ACL*
- Files with identical access rights: share parent's tree ACL
- Files with different access rights: *private ACL*

Tree ACL initialization and update

- Can be set manually or automatically
- Can be updated either statically or dynamically

Security analysis: Players

Principals

- Native principals (trusted), tenant principals (untrusted)

Clients

- Trusted entities that provide filesystem access to principals

Storage servers

- Trusted storage devices that store and return data

Metadata servers

- Trusted servers that manage filesystem namespace

Authentication servers

- Trusted servers which certify other players

Wire

- Transfers data between players

Security analysis: Possible attacks

Attacks on the wire

- Captured credential, Denial of Service
- Encrypted/signed credentials, message nonches to prevent replays

Attack on a client or tenant principal

- Attacker can access the principal's data
- Attack is confined within the principal's tenant

Attack by a revoked tenant

- FS revokes tenant access: tenant ACL is deleted

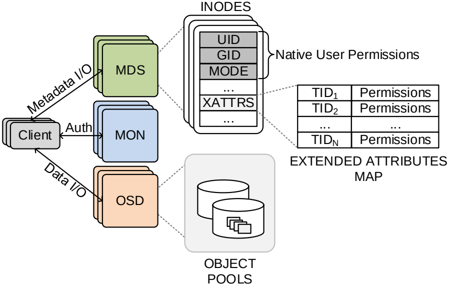
Attack on a native filesystem principal

- Attacker can gain complete access to the data of all tenants
- Secure these accounts!

Attack on tenants' data

- Reasons for which a provider is not trusted for critical data
- Tenants may externally apply data-protection techniques

Prototype Implementation



Overview

- Based on Ceph Version 0.61.4 (Cuttlefish)
- Main modified components: MDS, Client, Messages, Tools

Multitenant access control

- FS mount: Clients identify their tenants
- Client session limited to a single tenant more
- Tenant view: File permissions stored in EAs (C++ map) more
- Admin view: File permissions stored in regular Inode fields more

Experimentation environments

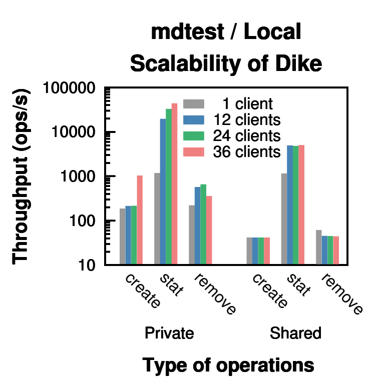
Local

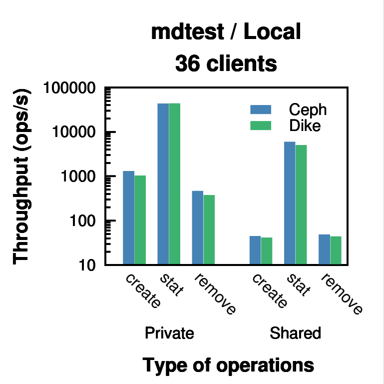
- amd64-based HP ProLiant DL140 G3 server nodes
- 250-500 GB, 7200 RPM HDs, 1 Gbps NET
- **MDS:** x1, 1 x Intel E5345, 6 GB RAM, Linux 3.9.3
- **OSD:** x3, 1 x Intel E5345, 3 GB RAM, Linux 3.9.3, XFS FS
- **MON:** x1, 1 x Intel E5345, 3, GB RAM, Linux 3.9.3
- **DOM0:** x6, 2 x Intel E5345, 4 GB RAM, Linux 3.5.5, Xen 4.2.1
- **Client:** x36, 1 VCPU, 512 MB RAM, Linux 3.9.3, bridged NET

Amazon Web Services (AWS)

- **m1.xlarge**: x3, 4 VCPU, 15 GB RAM, Linux 3.9.3
- **t1.micro**: x32, 1 VCPU, 615 MB RAM, Linux 3.9.3
- **c1.medium**: x1, 2 VCPU, 1.7 GB RAM, Linux 3.9.3

mdtest: Ceph vs Dike on local testbed





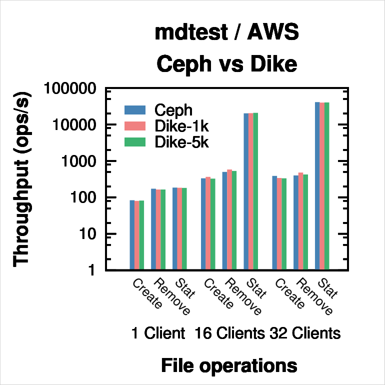
Configuration

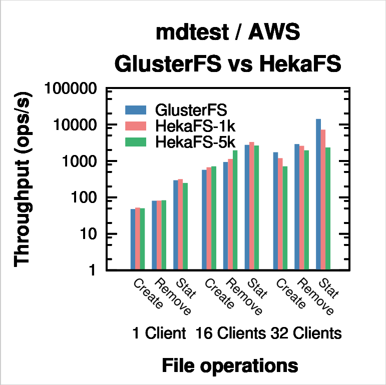
- 31104 created files & folders. Dike supports 36 tenants. 12 tasks

Dike overhead: 0-20%

- Mostly affected operation: create over a private folder

mdtest: Multitenancy overhead comparison on AWS





Configuration

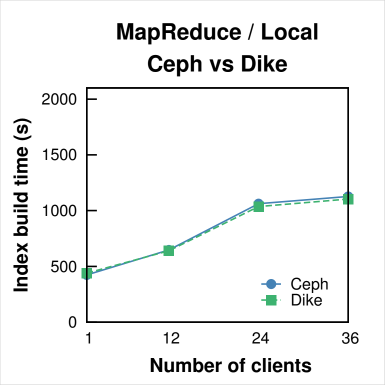
- 48000 created files & folders. t1.micro EC2 instances for clients
- 3 fileservers in total (m1.xlarge instances). 5 tasks/client

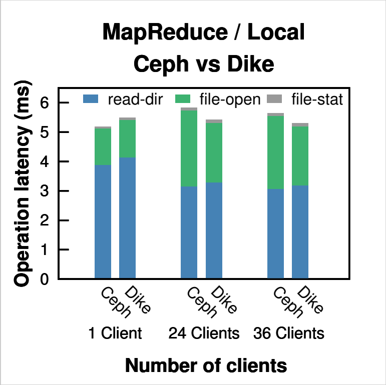
Dike: limited performance overhead compared to Heka

more

- Dike overhead: 12% for 1000 tenants, 14% for 5000 tenants
- Heka overhead: 49% for 1000 tenants, 83% for 5000 tenants

MapReduce: Ceph vs Dike on local testbed





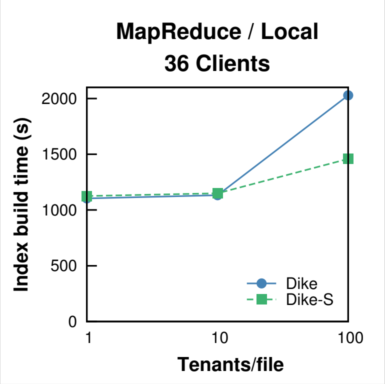
Configuration

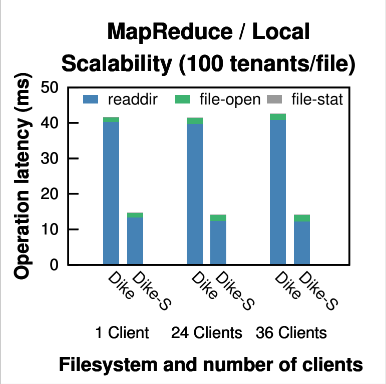
- Shared dataset: 78255 HTML files in 14025 folders, occupying 1 GB
- Reverse index: Generates the text index with links to the files

Dike overhead: 0-3.8%

- Single client: 3.8% overhead
- readdir latency: 7% higher when Dike is used

MapReduce: Long ACLs/ACL sharing on local testbed





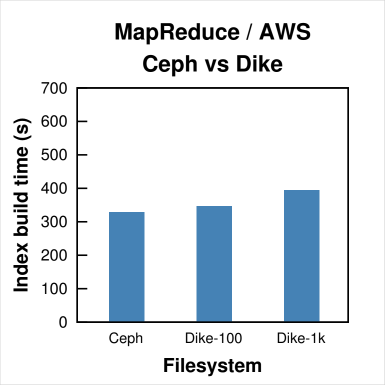
Configuration

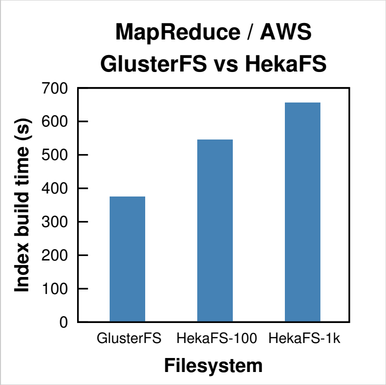
- Shared dataset: 78255 HTML files in 14025 folders, occupying 1 GB

Long ACLs: degrade system performance

- ACL sharing reduces index building time by 39%
- ACL sharing reduces readdir latency by 70%

MapReduce: Multitenancy overhead comparison on AWS





Configuration

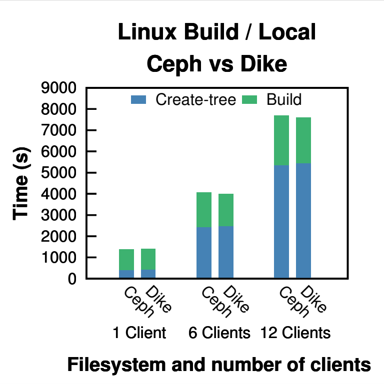
- Shared dataset: 78255 HTML files in 14025 folders, occupying 1 GB
- Fileservers: 3 in total (m1.xlarge), Client: 1 c1.medium instance

Dike: limited performance overhead compared to HekaFS

more

- Dike overhead: 5% for 100 tenants, 20% for 1000 tenants
- Heka overhead: 31% for 100 tenants, 75% for 1000 tenants

Linux build on local environment



Configuration

- Shared Linux 3.5.5 source. Accessible to private folders through links

Dike: negligible overhead

- Soft link creation: 2% with 12 clients, 4.5% with 1 client
- Kernel build: 0% with 12 clients, 0.7% with 1 client

Conclusions and future work

Per-tenant authentication servers and ACLs

- Tenants can manage their principals locally
- Identity isolation: Tenant principals/permissions in dedicated ACLs
- Avoidance of identity mappings and centralized directory

Namespace filtering

- Namespace isolation: Tenants can access a filtered view of the FS

ACL sharing

- ACL size limitation: Files with identical rights share parent's ACL

Performance

- Limited performance overhead
- Lower overhead/better scalability in comparison to ID mapping

Future work

- Weaker trust assumptions, further experiments, integration into a trusted virtualization environment

Tenant isolation in general

Hardware level

- Dedicated physical server per tenant
- Not scalable, wasted hardware resources, increased costs

Hypervisor level

- Shared hypervisor and separate VMs for each tenant
- Performance overheads, no sharing ability

Operating system level

- Shared server hardware and OS: kernel performs tenant isolation
- Low execution overheads but no sharing ability

Application level

- Shared server hardware, operating system, and server application
- Hard to achieve but cleanest way to isolate multiple tenants
- Enables sharing and high scalability

Implementation details: Client session

Client authentication

- Client authenticates to MON: session key encrypted with shared key
- Client uses session key to securely request ticket from MON
- Ticket: authenticates clients to MDSs and OSDs

Session initiation

- Clients use the ticket to initiate a new session with the MDS
- MDS receives an *MClientSession* message and returns a capability
- *MClientSession* message extended to contain the tenant ID
- MDS extracts the tenant ID and stores it in session state

back

Implementation details: Modifications

New methods to set and retrieve permissions of tenants/principals

Modifications to all FS functions related to permissions handling

Tenant View

- We use the tenant ID as a key to refer to a particular EA value
- We save the UIDs/GIDs/permissions into EAs
- We update the regular inode access control fields according to the UID/GID/permissions of the parent inode
- The client can not access the EAs that contain access control info

Admin View

- We directly update the regular inode access control fields

Capabilities

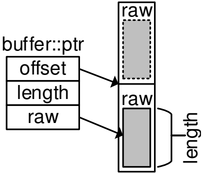
- Extended to contain the tenant ID
- Only sent to clients whose tenant has access to the file

back

Implementation details: Ceph structures

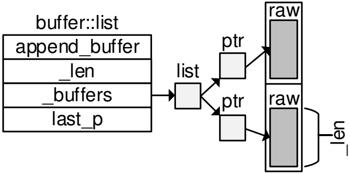
Structure: buffer/buffer::ptr

- In-memory data processing
- Data stored in buffer::raw objects
- Page-aligned memory
- buffer::raw can be accessed through a buffer::ptr pointer



Structure: buffer::list

- List of buffer::ptr pointers

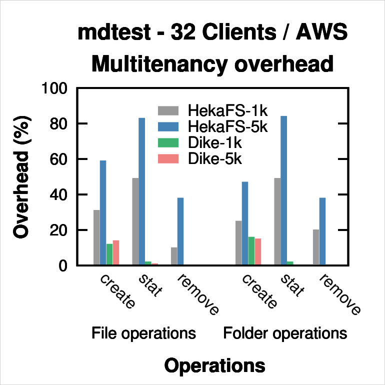


Extended Attributes

- C++ map structure (red-black tree)
- Each entry is a key/value pair
- The key is the name (string), the value is a buffer::ptr structure

back

mdtest: Dike vs Heka on AWS



Configuration

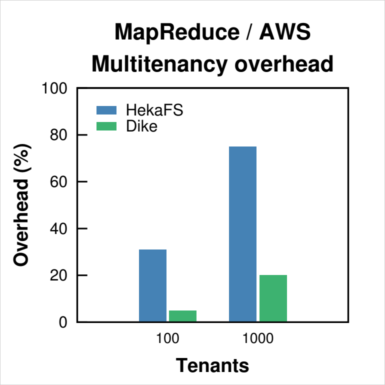
- 48000 created files/folders. t1.micro EC2 instances for clients
- 3 fileservers in total (m1.xlarge instances). 5 tasks/client

Number of tenants does not affect Dike

Heka adds a significant overhead of up to 84% to Gluster

back

MapReduce: Dike vs Heka on AWS



Configuration

- Shared dataset: 78255 HTML files in 14025 folders, occupying 1 GB
- Fileservers: 3 in total (m1.xlarge), Client: 1 c1.medium instance

Dike adds limited overhead to Ceph back

Heka adds a significant overhead of up to 75% to Gluster