Managing data reliability and integrity in federated cloud storage

Krzysztof Styrc

AGH University of Science and Technology, Kraków
kstyrc@gmail.com

Supervisor: Marian Bubak, PhD
Advisor: Piotr Nowakowski

Agenda

Objectives

Data in VPH-Share Cloud Platform:

- **sensitive** data on one or more cloud storage nodes (S3, Swift, . . .),
- data has to be **available** and **valid**.

The aims of this work:

1. **efficient** data validation mechanism in federated cloud storage,
2. implementation under various practical **limitations**.

Essentials aspects of this work:

- design and implementation of DRI REST WS enabling data validation,
- integration with the VPH-Share Cloud Platform environment,
- design of a bandwidth-efficient validation algorithm

Data integrity in cloud storage

Why data validation in cloud storage?

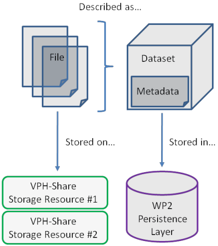
SLA guarantes 99.9%, if not met – credit return:
– *http://aws.amazon.com/s3-sla/*
– *http://www.rackspace.com/cloud/legal/sla/*
– *https://developers.google.com/appengine/sla*
Moreover, in replicated environment one can use other cloud provider while primary is unavailable or contains corrupted data.

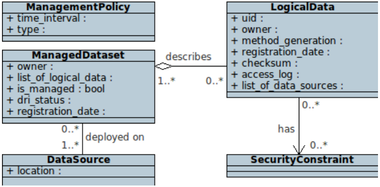
Recent problems with the *cloud* services:

- Gmail: erased emails, blocked accounts,
- Amazon S3: serving corrupted data, unavailable data,
- GoogleDocs: unauthorized data access,

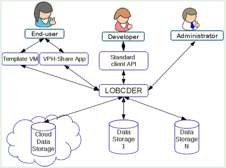
VPH-Share environment

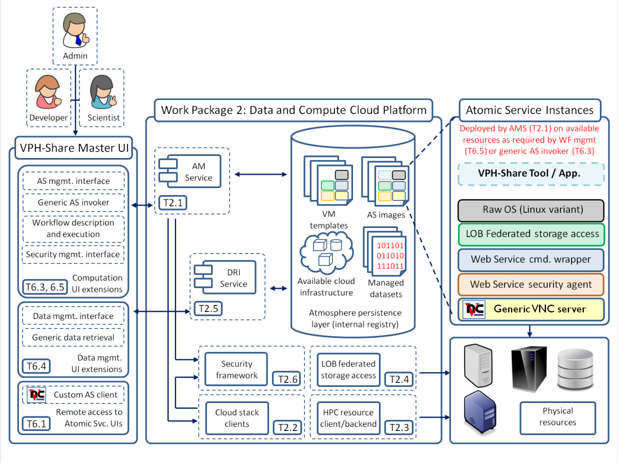
- **Atmosphere Internal Registry** (AIR) – metadata database of datasets, files, storage nodes, configuration, policy,
- data structure: *dataset* = *set of files*





Storage nodes – direct access to cloud storage nodes rather than LOBCDER layer (S3, Swift, . . .).





Functional requirements:

- periodic and on-request validation,
- data replication,
- user notification about integrity errors.

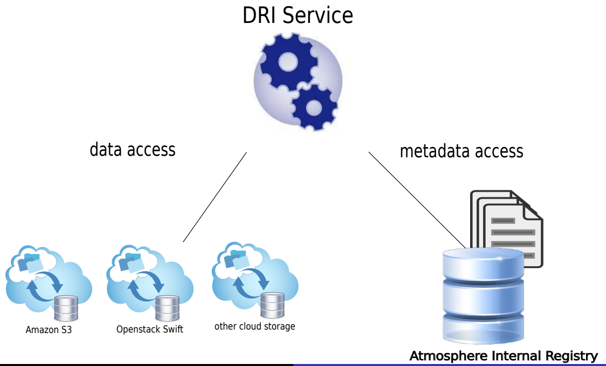
Nonfunctional requirements:

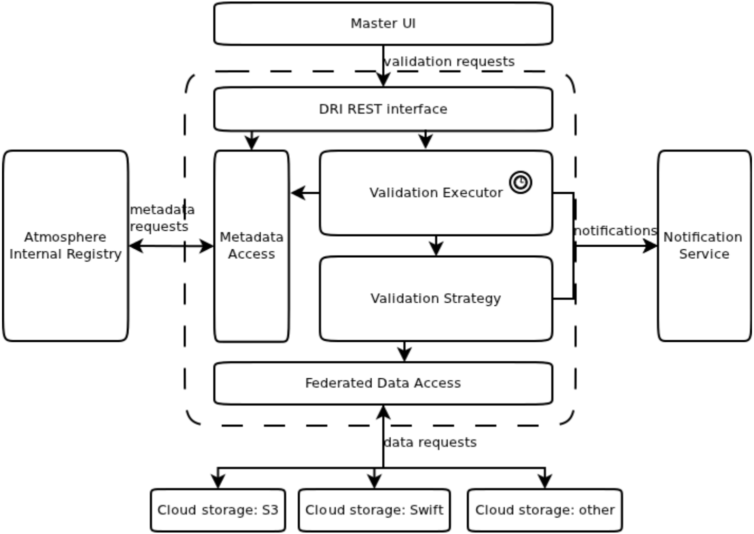
- **bandwidth-efficient** validation mechanism,
- scalablility,
- configurability.

DRI architecture

DRIService

+ registerDataset(dataset : ManagedDatasetDescription) : ManagedDatasetID
+ unregisterDataset(id : ManagedDatasetID)
+ replicateDatasetToResource(id : ManagedDatasetID, source : DataSourceID)
+ dereplicateDatasetFromResource(id : ManagedDatasetID, source : DataSourceID)
+ datasetChanged(id : ManagedDatasetID, dataset : ManagedDatasetDescription)
+ validateDataset(id : ManagedDatasetID) : Message
+ setManagementPolicy(policy : ManagementPolicy)
+ getManagementPolicy(id : ManagedDatasetID) : ManagementPolicy




Master UI
validation requests
DRI REST interface
Atmosphere Internal Registry
metadata requests
Metadata Access
Validation Executor
Validation Strategy
notifications
Notification Service
Federated Data Access
data requests
Cloud storage: S3
Cloud storage: Swift
Cloud storage: other

Cloud storage characteristics

- hierarchical structure: *account / container / object*,
- typical operations: *list, get, put, delete, update, ...*,
- REST API goes toward CDMI/OCCI standards:
  - *http://docs.amazonwebservices.com/AmazonS3/latest/API/*
  - *http://docs.openstack.org/api/openstack-object-storage/1.0/content/*
  - *http://docs.rackspace.com/files/api/v1/cf-devguide/content/index.html*
- SLA quality, best quality–price ratio, scalability, ...,

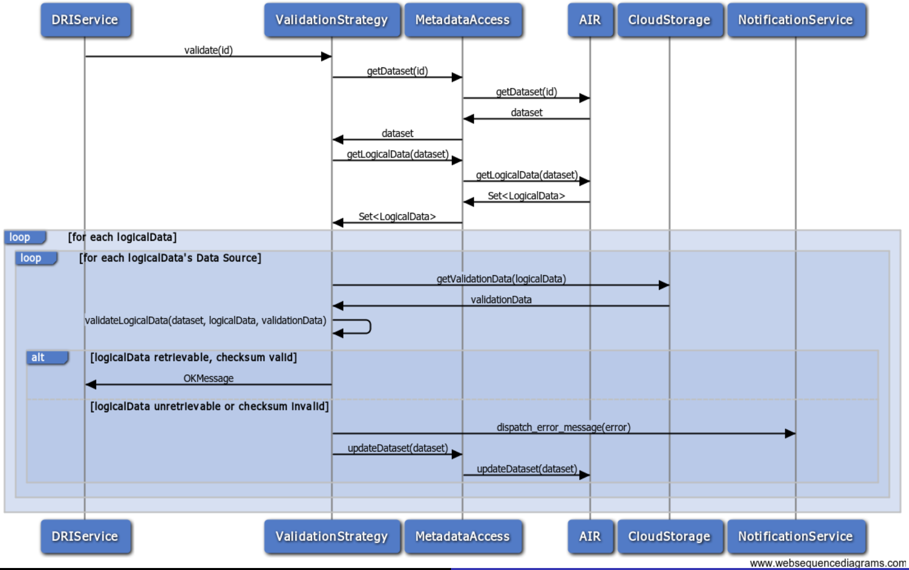
No support for HTTP Multi-Range requests:

- new request for every chunk,
- flooding the cloud storage,
- really bad performance $\rightarrow$ RTT for every chunk.

Validation algorithm pseudocode

```
def validate_dataset(dataset id) {
    dataset = get_dataset_metadata(id);
    files = get_dataset_files(dataset);
    for file in files {
        for data_source in dataset.get_data_sources() {
            data = get_validation_data(file, data_source);
            if data == null # unable to retrieve data
                dispatch_error_message(file, data_source);
            result = validate_data(data, file, dataset);
            if not result # file is invalid
                dispatch_error_message(file, data_source);
        }
    }
}
```

Validation algorithm sequence diagram



Implementation technologies

- **Jersey API** – JAX-RS reference implementation, REST communication
- **JClouds** – very good cloud operability library,
- **Quartz** – tasks execution and scheduling library,
- **Guice** – dependency injection and modularity library,
- **Maven** – build + dependencies tool,
- **Apache Tomcat** – server.

Validation efficiency

Significant size of data stored in federated cloud storage makes whole-file checksum-based validation completely inefficient. Additionally, occupying substantial portion of network bandwidth is highly undesirable.

For example:

typical dataset $\approx$ 10 GB $\rightarrow$ goal: reduce to $\approx 10\%$

Can we do better?

Generally yes, but lots of practical limitations:

- cloud storage limitations,
- files cannot be modified in VPH-Share.

Promising approaches:

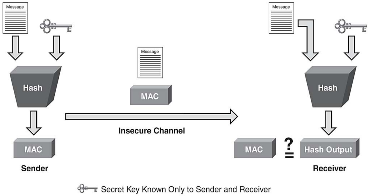
1. Proofs of Retrievability (PoR),
2. Data integrity proofs (DIP).

Classical integrity mechanisms

- **Hash functions** – for detecting data corruption (MD5, SHA1, . . .),
- **Error Correcting Codes (ECC)** – for correcting small parts of data,
- **Message Authentication Codes (MAC)** – hash + auth_key for secure authentication.

Validation mechanism steps:

1. setup: compute and store,
2. validate: compute + compare.



Proofs of Retrievability scheme

Setup:

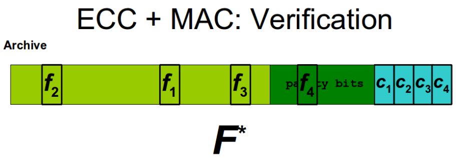
1. divide into chunks,
2. compute ECC + MAC,
3. $\frac{F^*}{F} \approx 102 - 115\%$.

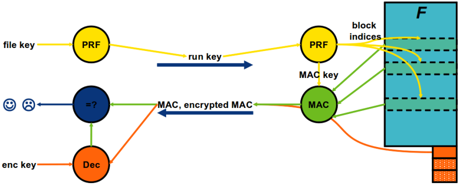
Validation:

1. challenge–response rounds:
2. more rounds → higher probability,
3. more rounds → higher bandwidth.

Problems:

1. we cannot modify stored file,
2. data is served by LOBCDER, ECC useless.





Data integrity proofs scheme

Setup:

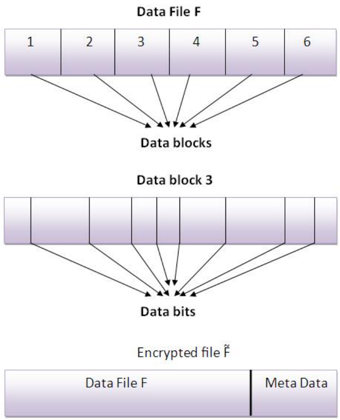
1. divide into k chunks,
2. pick m bits in every chunk,
3. compute MAC and store.

Validation:

1. compute as in setup,
2. compare with original MAC.

Problems:

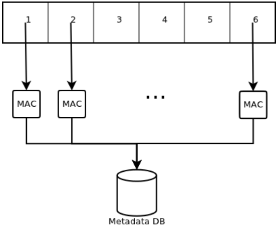
1. cannot query cloud storage bit by bit,
2. cannot append to stored file.



Our solution – mixed PoR + DIP

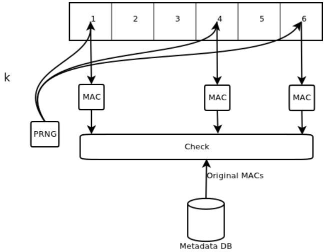
Setup:

1. divide into chunks of size S,
2. compute MAC of every single chunk and store.



Validation:

1. pick sequence of k random indexes (Mersenne-Twister),
2. get k chunks data,
3. compute and verify with originals.



Solution features:

- configurable accuracy–overhead ratio,
- practical feasibility.

Work status:

- working periodic and on-request validation,
- integrity errors notifications page,
- task scheduling + configurability,
- **testing the implementation of our solution**,
- **thesis writing start**.

Conclusions:

- cloud storages have their usability limitations,
- remote data validation – not a trivial task,
- efficient validation only with high probability.

Future work:

- integrate LOBCDER and DRI tools,
- watch future versions of CDMI/OCCI standards for significant impovements.