Crystal: Software-Defined Storage for Multi-tenant Object Stores

Raúl Gracia-Tinedo, Josep Sampé, Edgar Zamora, Marc Sánchez-Artigas, Pedro García-López (Universitat Rovira i Virgili, Spain)

Yosef Moatti, Eran Rom (IBM Research-Haifa, Israel)
A LESSON FROM A PALM TREE
A lesson from a palm tree

• When the weather is good, all the trees enjoy 😊
A lesson from a palm tree

- But weather conditions may change dramatically…
  And, when the storm comes…

http://crystal-sds.org
https://github.com/Crystal-SDS
A lesson from a palm tree

• Some trees get through a storm better than others.

• One of the key properties for a tree to survive a storm is being flexible.
A lesson from a palm tree

- This work aims to make **object stores flexible** for adapting the system to **changing requirements**.

  - **Flexibility**
    - Programmability
    - Extensibility
    - Automation
    - Control
CONTEXT: SOFTWARE DEFINED OBJECT STORAGE
What is Software-Defined Storage?

• Hey! What does Software-Defined Storage mean?

“Software-defined storage (SDS) is a new term for data storage software that provides policy-based provisioning and management of data storage independent of the underlying hardware.”

(Wikipedia)

• SDS: Automation, management, optimize workloads,…
SDS Concepts & Works

• SDS systems **decouple control/data planes:**
  – *Control plane:* Definition of policies, intelligence
  – *Data plane:* Enforcement of policies, manage data flows

• SDS systems in the literature:
  – **IOFlow** (*SOSP’13*): BW control/middleboxes, targets **file-system**
  – **Retro** (*NSDI’15*): **Resource management**, guarantee SLOs
  – **sRoute** (*FAST’16*): **Network-like model** to treat flows (sSwitches)

• We focus on **object storage** (OpenStack Swift):
  – **Policy-based redundancy** models in Swift (from Kilo version)
  – Companies like **SwiftStack** sell automation, provisioning, and metering services.

• **But, can we go a step further? And, why?**
PROBLEM: ADVANCED STORAGE MANAGEMENT
What if…

Tenant 1: Stores compressible data

Tenant 2: Stores non-compressible data

Can we compress only T1’s data to save storage space?

http://crystal-sds.org
https://github.com/Crystal-SDS
Or, what happens if…

Tenant 1:
SLO=30MBps

Tenant 2:
SLO=60MBps

Can we enforce service-level objectives on tenants?

http://crystal-sds.org
https://github.com/Crystal-SDS
Sadly, Swift lacks from flexibility…
CRYSTAL
Design Principles of Crystal

• Crystal is a **SDS framework to solve storage management problems** in object stores (OpenStack Swift):
  – It is **not** an ad-hoc solution to a particular problem!

• Crystal decouples **control/data planes**:

  ![Control Plane](#) Policies, Controllers

  ![Data Plane](#) Filters, Triggers
Crystal Design (Data Plane)

- **Filter**: Data transformation executed on object requests. E.g., compression, caching, encryption, bandwidth allocation...

- **Two types of filters**:  
  - **OpenStack Storlets** (isolated execution)  
  - **Native filters** (non-isolated execution)

**Filters are pluggable at runtime!!**
Crystal Design (Data Plane)

- **Inspection triggers**: Information of a particular aspect of the system operation to trigger the execution of filters.
  - **Object metadata**: object size, type,…
  - **Workload metrics**: E.g., requests/sec, CPU load, object contents,…

Metrics are pluggable at runtime!!

PUTS_SEC

Workload metric process

PUTS_SEC

Swift Proxy

Swift Storage Node

Workload metrics framework

Monitoring info

http://crystal-sds.org

https://github.com/Crystal-SDS

@iostackproject
Crystal Design (Control Plane)

• **Controller**: Algorithm that receives as input workload metrics to manage the execution of filters.

Layer of distributed controllers  Workload metric processes

• **We provide two types of controllers:**
  – **Automation controllers**: Simple activation rules (*self-generated*).
  – **Global controllers**: Global visibility and coordination of a filter at the data plane (*user-defined*).
Crystal Design (Control Plane)

- **IFTTT-like policies**: Crystal provides administrators with a simple “If-This-Then-That”-like DSL

<table>
<thead>
<tr>
<th>FOR</th>
<th>[TARGET]</th>
<th>WHEN</th>
<th>DO</th>
</tr>
</thead>
<tbody>
<tr>
<td>P1</td>
<td>TENANT T1</td>
<td>OBJECT_TYPE=DOCS</td>
<td>SET COMPRESSION WITH TYPE=LZ4, SET ENCRYPTION</td>
</tr>
<tr>
<td>P2</td>
<td>CONTAINER C1</td>
<td>GETS_SEC&gt;5 AND OBJECT_SIZE&lt;10M</td>
<td>SET CACHING ON PROXY TRANSIENT</td>
</tr>
<tr>
<td>P3</td>
<td>TENANT T2</td>
<td></td>
<td>SET BANDWIDTH WITH GET_BW=30MBps</td>
</tr>
</tbody>
</table>

- The vocabulary of the DSL can be expanded at runtime:
  - By adding new **filters**
  - By adding new **workload metrics and triggers**
EXPERIMENTAL RESULTS
Setup

• 13 nodes cluster:
  – 1 Controller: controllers, metadata, messaging, authentication, Swift proxy (Dell PowerEdge 420, 32GB RAM, 1TB HDD)
  – 3 nodes to execute workloads (Dell PowerEdge 420, 32GB RAM, 1TB HDD)
  – 2 Swift Proxies (Dell PowerEdge 320, 28GB RAM, 1TB HDD, 500GB SSD)
  – 7 Storage nodes (Dell PowerEdge 320, 16GB RAM, 2x1TB HDD)

• Switched 1GBit Ethernet

• OpenStack Swift Kilo version
Storage Automation

Control Plane

```plaintext
FOR CONTAINER C1 WHEN GETS_SEC > 5
DO SET CACHING ON PROXY
```

Data Plane

Container C1

```
GETS_SEC > 5
```

Swift Proxy

Swift Storage Nodes

Metadata store

DSL Compiler

Crystal APIs

GETS_SEC

Monitoring info

GETS_SEC

GETS_SEC

Storage Automation

- Oscillatory workload with high locality (0 to 10 PUT/GET per sec.).
- Simple LRU caching filter exploiting proxy SSDs.
- Enforce caching under higher loads (GET_SEC>5)
  - Reads are 30% faster (in median) when caching is activated.
  - Writes are almost not affected.

http://crystal-sds.org
https://github.com/Crystal-SDS
**Bandwidth Differentiation**

**Control Plane**

FOR TENANT T1 DO SET BANDWIDTH WITH GET_BW=30 ON STORAGE_NODE

**Data Plane**

Swift Proxy

Swift Storage Nodes

---

DSL Compiler

Crystal APIs

Metadata store

Monitoring info
Bandwidth Differentiation

- **3 ssbench tenants** executing 128MB GETs against the cluster.
- **Bandwidth throttling filter** (deployed at storage nodes).
- **2 global controllers** (GET BW, Replication BW)
- Until replication traffic is controlled, SLOs are not achieved.

It is very difficult to guarantee SLOs until replication bandwidth is controlled.
CONCLUSIONS
Conclusions

• Many object stores are hard to adapt to new requirements.

• We presented Crystal, the first SDS architecture for object stores:
  – Control plane: Policies (IFTTT-like), Controllers (distributed algorithms)
  – Data plane: Filters (compute on objects), Triggers (metrics, metadata)

• We demonstrated the extensible design of Crystal:
  – New storage automation policies (compression, caching,…).
  – A global bandwidth control filter.

• The framework opens the door to investigate new storage filters and control algorithms.
QUESTIONS?
THANK YOU!

http://crystal-sds.org

https://github.com/Crystal-SDS
Overheads

- **Metadata access overhead:**
  - 3.9% 1MB

- **Isolated flow interception:**
  - 15.7% 1MB
  - 5.7% 10MB

- Performance may also depend on data content & access patterns.

---

URLs:

- [http://crystal-sds.org](http://crystal-sds.org)
- [https://github.com/Crystal-SDS](https://github.com/Crystal-SDS)

---

PUT Execution Time (seconds)