Methods and Tools for Federating Data

Spiros Koulouzis

University of Amsterdam
Institute for Informatics
Systems Network Engineering group

03 November 2015
Table of contents

Motivation for Data Federation

Challenges

Objectives

Data Federation Architectures
  Centralized Architecture
  Distributed Architecture
  SDN-Aware Architecture

Performance Evaluation

Lessons Learned from VPH-Share

Conclusions and Future Work
Motivation: Distributed Heterogeneous Resources

- Researchers need effective means to access, exchange and share data
- Many research groups have data “locked up” in their servers
- Computation resources are distributed
- Horizontal scaling (scale out) is the mainstream approach to solve data-intensive problems
- Data often reside in different locations than the computation
- Moving computation to the data is not always possible

---

Group, L. R. D. W., et al., 2013. LERU roadmap for Research Data. ADVICE PAPER 14, 36
Motivation: VPH-Share

- Virtual Physiological Human (VPH) aims at understanding the physiological processes in the human body
- The VPH-Share project built and deviled infrastructure used by medical applications
- Several providers made medical data available
- Several heterogeneous storage resources on different locations
- Many workflows (VMs) and users from several locations needed data
- Data access should be intuitive for users and applications

Federation Challenges

- Data federation attempts to aggregate data from different sources & provide to applications and users a unified view of data
- The heterogeneity of data access technologies makes data sharing and federation difficult
- Handling many client implementations to read and write data from different sources is not effective
- In cloud storage each provider uses it’s own API
- With the use of proprietary APIs, standards are imposed by vendors and can be incompatible with previous versions
- Storage providers often share storage only if specific requirements are met (type of data, bandwidth limits, etc.)

URL http://doi.acm.org/10.1145/304181.304209

Loshin, D., 2013. Integrating structured and unstructured data. TDWI Research
Objectives

- Research infrastructures (RIs) have to be loosely connected with flexible, scalable and transparent architectures.
- Sharing and accessing data is possible by providing easy to use architectures that can transparently handle heterogeneous data resources.
- The main objective is to provide efficient, scalable, flexible and transparent architecture for sharing and accessing files and create smart data processing RIs that can support diverse scientific workflows.
Building a Data Federation System

- Centralized Architecture: How to loosely-couple storage resources
- Distributed Architecture: Improve file availability and load-balance
- SDN-Aware Architecture: Improve execution times of workflows
Centralized Architecture

- Large OBJect Cloud Data storagE fedeRation (LOBCDER) is divided into three main layers: frontend, resources and backend

URL http://dx.doi.org/10.1007/978-3-642-54420-0_2
Centralized Architecture

- **Frontend**: WebDAV interface. Gives access to metadata & files

- **Resource**: Maps logical to physical files & executes tasks (replicate, delete, etc.)

- **Backend**: Abstraction to uniformly access physical storage resources

- LOBCDER implements a number of replication policies e.g. full redundancy, demand, etc.

- Storage resources can be public or private

- Implements “umount” for storage resources
Distributed Architecture

- Distributed LOBCDER is composed of two main components
  - Master
  - Workers
Distributed Architecture

- The LOBCDER master is distinguished between R and RW master
- They can be replicated across several machines
- LOBCDER workers load-balance requests, maintain file availability
- Translate protocols of storage backends into HTTP
- Stateless composed of two layers: the frontend and backend
Distributed Architecture

- To download a file the master selects a worker to transfer the data based on load-balancing algorithms.
- The user is redirected (HTTP code 302) to the selected worker.
- The worker gets information about the physical data location from the master.
- The worker relays the data to the user.
- While a worker sends a file to a user it adds it to its local cache.
SDN-Aware Architecture

- LOBCDER must select between a number of workers based on the quality of the path to data consumers.
- We use software defined networking (SDN) to make the worker selection and dynamically route data.
- SDN removes the control plane from network devices and places it into an SDN controller.
- In SDN the data plane is also responsible for monitoring local information and gathering statistics.


SDN-Aware Architecture

- Used an agent-based system (NEWQoSPlanner) to aggregate information from the network and control SDN controllers.
SDN-Aware Architecture

- A consumer cns requests data which can be downloaded from multiple data sources d
- We try to solve the multiple source shortest path (MSSP) problem with cost function:

\[
c_{d_i,cns} = c_{d_i,s_i} + \sum c_{s_i,s_l} + c_{s_{cns,cns}}.
\]
Distributed Architecture: Experimental Setup

- One LOBCDER master & 16 workers on 16 nodes on DAS-3 (Distributed ASCI Supercomputer)
- Three storage resources with files replicated on all of them
- Conducted experiments using 128, 256, 512 and 1024 users requesting the same file simultaneously
- Using Round-robin for worker selection
- On each test 1, 2, 4, 8 and 16 workers (each on a separate node) were used respectively
Distributed Architecture: Average User Session

- One session is the time between a GET request and an OK response.
- More workers results in lower session time.
Distributed Architecture: User Throughput

- 1024 users simultaneously request the same file
- Measure of how many users per second LOBCDER servers
SDN-Aware Architecture: Experimental Setup

- Used a Montage workflow trace to test SDN-Aware architecture (which worker to select)
- Workflow tasks don’t perform any execution only file requests
- **Time between file requests is almost zero**
- Consumers 1, 2 and 3 execute one of the three strands of the workflow
- Consumer 1 joins images
SDN-Aware Architecture: Experimental Setup

- We run the workflow using 8 VMs, 3 open Vswitches, and 12 links.
- Storage resources are on worker nodes.
- Two worker selection algorithms: i) Round-robin (not using SDN), ii) SDN-Aware.
- Two experiments: i) exclusive use of network (no traffic), ii) nodes 1 and 3 generate traffic.
SDN-Aware Architecture: Execution time for Empty Network

- Execution times of each request
- At 1st stage of the execution SDN-Aware architecture selects best worker
- At 2nd stage we have intermediate files

- Intermediate files are not replicated on all workers fast enough
- If execution time was added on workflow tasks there would be more time to replicate files
- Using SDN-Aware architecture to select workers the execution time was 9.5 minutes with round-robin 11.6 minutes
SDN-Aware Architecture: Execution time for Busy Network

- Execution time with workers 1 and 3 generating additional traffic
- SDN-Aware is able to identify the least cost path

- Using SDN-Aware architecture to select workers the execution time was **12.3 minutes** with the round-robin **32.1 minutes**
Lessons Learned from VPH-Share

- LOBCDER served more than 5 million requests over 24 months
- We logged these requests and analyze them to answer some questions:
  - What should be optimized?
  - Where to deploy Workers?
  - Is Caching going to help?
  - Is prefetching going to help?
What should be optimized?

- Measured the total time LOBCDER spent on serving requests
- Counted the type of requests served by LOBCDER
  - GET requests make for 38% of the total time spent on all requests (outer ring)
  - GET requests represent 5% of total number of requests made (inner ring)
Where to deploy Workers?

- Counted how many GET requests came from how many locations (cities)
- Constructed a cumulative percentage
- 96% of GET requests originated from 10% of the locations
Replayed all requests from 4 locations: Krakow, Sheffield, Oxford and Amsterdam (94% of GET requests)

Measured hit ratios for cache replacement algorithms
Caching: Why not everywhere?

<table>
<thead>
<tr>
<th>Location</th>
<th>Unique Files Requested, %</th>
<th>Maximum Hit Ratio, %</th>
</tr>
</thead>
<tbody>
<tr>
<td>Krakow</td>
<td>57.0</td>
<td>43.0</td>
</tr>
<tr>
<td>Sheffield</td>
<td>53.0</td>
<td>47.0</td>
</tr>
<tr>
<td>Oxford</td>
<td>82.9</td>
<td>17.1</td>
</tr>
<tr>
<td>Amsterdam</td>
<td>6.6</td>
<td>93.4</td>
</tr>
</tbody>
</table>

- Analyzed GET request from each of four locations. Counted unique files requests.
- The maximum hit ratio that can be achieved depends on the number of unique files requested.
Prefetching: Is it going to help?

- Counted how many files were contained on how many folders
- Constructed a cumulative percentage
- For Oxford 50% of the requested files are in 158 folders
- For Amsterdam 50% of the requested files are in 2 folders
Prefetching: How fast should it be?

- Counted how much time was passed between two consecutive requests
- Constructed cumulative probability of the time that passed between two consecutive GET requests
- The probability of Amsterdam making two requests within a second or less is 0.5. For Oxford it is 0.9
Conclusions

- LOBCDER is data federation architecture
- It uses virtualization to easily integrate heterogeneous storage resource
- Doesn’t require from storage owners to install additional software
- It is a scalable distributed file system with the use of workers that relay GET requests and can be deployed almost anywhere
- What can we optimize? It depends
Future Work

- We should automatically identify when, where and how many workers we need
- Look for trends in requests
- Further analyze requests & publish the DB (1.3 GB and counting)
Publications

