IJDC | *General Article*

OpenStack Swift: An Ideal Bit-Level Object Storage System for Digital Preservation

Guanwen Zhang
University of Alberta Library

Kenton Good
University of Alberta Library

Weiwei Shi
University of Alberta Library

Abstract

A bit-level object storage system is a foundational building block of long-term digital preservation (LTDP). To achieve the purposes of LTDP, the system must be able to: preserve the authenticity and integrity of the original digital objects; scale up with dramatically increasing demands for preservation storage; mitigate the impact of hardware obsolescence and software ephemerality; replicate digital objects among distributed data centers at different geographical locations; and to constantly audit and automatically recover from compromised states. A realistic and daunting challenge to satisfy these requirements is not only to overcome technological difficulties but also to maintain economic sustainability by implementing and continuously operating such systems in a cost-effective way. In this paper, we present OpenStack Swift, an open-source, mature and widely accepted cloud platform, as a practical and proven solution with a case study at the University of Alberta Library. We emphasize the implementation, application, cost analysis and maintenance of the system, with the purpose of contributing to the community with an exceedingly robust, highly scalable, self-healing and comparatively cost-effective bit-level object storage system for long-term digital preservation.

Keywords: Digital objects, long-term digital preservation, bit-level digital preservation, OpenStack Swift, physical infrastructure, object storage.

Submitted 10 July 2021 ~ *Accepted* 9 September 2022

Correspondence should be addressed to Guanwen Zhang, University of Alberta. Email: guanwen@ualberta.ca

The *International Journal of Digital Curation* is an international journal committed to scholarly excellence and dedicated to the advancement of digital curation across a wide range of sectors. The IJDC is published by the University of Edinburgh on behalf of the Digital Curation Centre. ISSN: 1746-8256. URL: http://www.ijdc.net/

http://dx.doi.org/10.2218/ijdc.v17i1.782
DOI: 10.2218/ijdc.v17i1.782

Introduction

Driven by technological advancement, data and information are mostly produced in digital form in the current information age; they are inherently fragile and volatile, and at a high risk of loss. To prevent the potential loss and inaccessibility of valuable digital assets, long-term digital preservation (LTDP) has become a necessity rather than an option, and has found widespread applications in libraries, archives and museums (Giaretta, 2011; Lu & Pan, 2010; Myntti & Zoom, 2019). The more advanced the information technologies that have evolved, the more digital data they generate, and the more demands emerge to preserve digital data of historical and future value. LTDP, undoubtedly a daunting task with great complexity, is a synergy of a wide variety of factors: technological, economical, legal, financial, organizational and managerial (Rieger, 2018). From the perspective of technology, it relies on the ability to preserve digital objects in bit streams, which is often referred to as bit-level digital preservation and constitutes an essential building block of LTDP. Hence, sound, cost-effective, and sustainable bit-level digital preservation infrastructures are vital to ensure the success of LTDP.

However, there exist a multitude of challenges for bit-level digital preservation, which have been well documented by the scientific community (Haughton, 2016; Hedstrom, 1998; Rosenthal, 2005, 2010). The first challenge is the scalability of the storage capacity with respect to the massive growth rate of digital data. Rosenthal reported that the average annual growth rate of data creation is some 60%; that of bit density of storage media is 20% or so; and that of data center budgets is in the vicinity of 2% (Digital Preservation Handbook, 2015; Rosenthal, 2014). The striking disparity inevitably leads to unnecessary exclusion of some digital objects from being preserved by the preservation infrastructure due to limited storage capacity. Secondly, an infrastructure, effective and efficient in preserving a smaller number of digital objects, might not scale up well with a large volume of digital objects, degrading the performance of auditing, accessing, and fixity checking, and rendering the preservation infrastructure less reliable. Thirdly, hardware deterioration and eventual failure are unavoidable. For instance, hard disks--employed to store the digital objects in bit streams--have a limited life span; they are susceptible to complete failure at times that are hard to predict (Kamarthi, Zeid, & Bagul, 2009). A bit-level digital preservation system should have the resilience to tolerate and recover from such failures without performance degeneration, service disruption, or worst of all, data loss. Fourthly, LTDP makes it imperative to maintain authenticity and integrity of the preserved digital objects, which necessitates constant auditing, frequent fixity checking and automatic recovery from data abnormalities. These are onerous tasks demanding efficient algorithms and sufficient computer processing power. Moreover, power failure, water leakage, fire and other natural disasters add more complexities to the challenges of bit-level digital preservation infrastructure (Rosenthal, 2005). It is highly desirable for such an infrastructure to be distributed and networked so that multiple copies of the same digital objects can be replicated among different data centers situated at separate geographical locations (Digital Preservation Handbook, 2015). As a result of assiduous planning with replication, distribution and interconnectedness in mind, a complete failure of one data center will not disrupt the accessibility of preserved digital objects, let alone cause data loss.

Over the last few years, OpenStack Swift--an open-source cloud object storage platform -- has attracted the attention of members of the digital preservation community because it offers a suite of desired features that can not only meet the requirements and overcome the challenges for long-term bit-level digital preservation, but also offers the digital preservationists a cost-effective solution (Arnold, 2014). The adoption of OpenStack Swift as a bit-level preservation platform has been gaining momentum. It has been at work at the University of Alberta Library (UAL) since 2013. Scholars Portal has capitalized on OpenStack Swift to provide LTDP services for members of the Ontario Council of University Libraries (OCUL) and non-members as well through the Ontario Library Research Cloud (OLRC) (Askey & Ruest, 2016). Many digital

preservation software and tools have extended or are in the process of extending their support for the integration of OpenStack Swift into their storage layers. Most typical examples include DuraCloud (Kimpton & Payette, 2010), Archivematica (Goodchild & Hurley, 2019), Dataverse (Durand, 2020), and Fedora Commons (Wilcox & Wenraub, 2017), to name a few. David Rosenthal (2013), co-founder of the LOCKSS digital preservation system, conducted a cost-effectiveness study by integrating LOCKSS boxes with a virtual storage box in Amazon cloud environment, and concluded that "[public] cloud storage is not even close to cost-competitive with local disk storage for long-term preservation purposes in general, and LOCKSS boxes in particular." Breeding (2012, p34) indicates that cost advantages for local storage become more pronounced in comparison with public cloud storage options. Using the 140TB of storage underlying the Vanderbilt Television News Archive as example, it would cost $16,560 per month, or $198,720 per year, had the Amazon S3 storage been used. To be cost-effective, OpenStack Swift can be deployed as an on-premises private cloud storage system for LTDP by leveraging its compatibility with commodity-grade hardware and HDDs.

However, there is a lack of systematic presentation of the use of OpenStack Swift by the community as a cloud-based digital object storage for the purpose of LTDP. The objective of this paper is to fill the void with a systematic introduction of OpenStack Swift: what LTDP features it has; how it can be used as storage for LTDP; and how it can be implemented, operated and maintained in the face of budgetary challenges. A case study of the usage of OpenStack Swift at UAL is presented, together with cost analysis of the competitiveness of on-premises OpenStack Swift storage with respect to Azure cloud. These are described with more details in the following three sections: OpenStack Swift; Implementation; and A Case Study. The final section of the paper is dedicated to conclusion and discussion.

OpenStack Swift

OpenStack Swift is an open-source cloud-based platform for object storage which is intended to run on any commodity hardware compatible with Linux or Windows Operating Systems (Arnold, 2014). It originated from a joint effort between NASA and RackSpace in their pursuit of a reliable storage system to store and preserve massive unstructured data. It is a software product under the umbrella of OpenStack, a large cloud computing platform, but can be run standalone.

Architecturally, OpenStack Swift usually consists of a node running the proxy service, a node running the authentication service, and a set of storage nodes running a suite of storage related services, which is schematically illustrated in Figure 1 (Kapadia, Varma, & Rajana, 2014; OpenStack Swift Architecture (OSA).[1] A 'node' here is interchangeable with a computer server: virtual or physical. A proxy node is the gateway for a client to persist or access digital objects on the storage nodes. An authentication node grants or denies access permission to a client depending on the result of credential verification. To scale up with load, enhance performance, and eliminate a single point of failure, multiple authentication and proxy nodes can be deployed with a load balancer. Additional storage nodes can be added to the cluster dynamically, or existing storage nodes can be removed if needed, making the storage capacity horizontally scalable without sacrificing performance. The storage nodes are distributed and interconnected, and grouped into geographic 'Regions'. Each 'Region' can have one or more 'Zones.' A 'Zone' is a failure boundary; it can be as large as a data center; it can be a rack of servers; or it can be as small as a single hard disk. The failure of one or more 'Zones' should not affect the functionalities and performances of the system. Each 'Zone' has one or more storage nodes, and each node can have as many hard drives as it can power. The above arrangement gives rise to an architecture with a hierarchy of four-tier failure domains: hard disks, storage

[1] OpenStack Swift Architecture (OSA) https://www.swiftstack.com/docs/introduction/openstack_swift.html

nodes, zones, and regions, offering tremendous tolerance of hardware or data center failure.

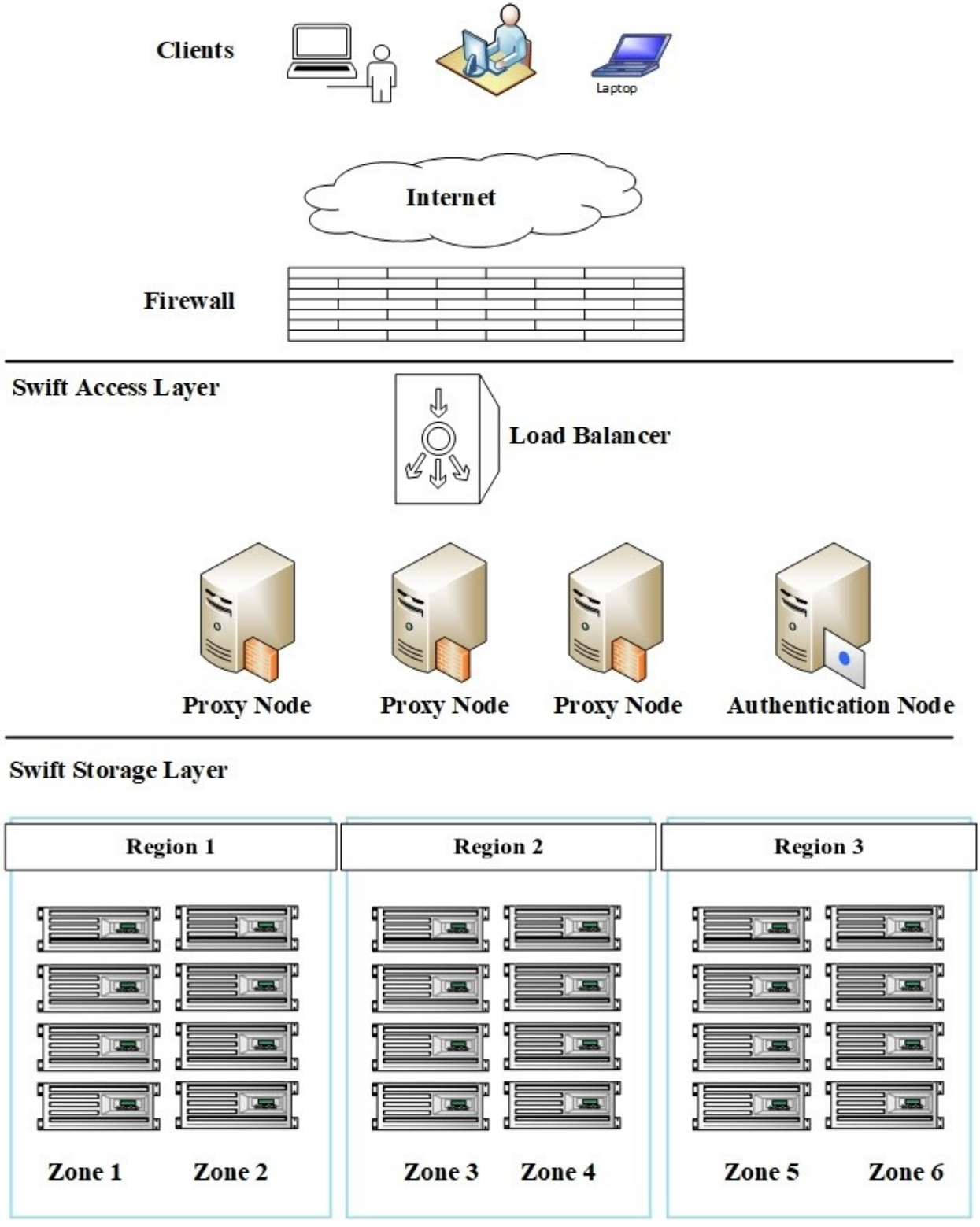

Figure 1: A schematic diagram of the deployment architecture of OpenStack Swift.

In operation, a digital object is preserved with a primary copy and at least two replicas; the number of replicas is configurable. If there are three or more regions, the primary copy and its replicas are distributed across regions, avoiding co-location at region level. The failure of a

region does not impact the end users. However, if OpenStack Swift is deployed with only one region, a digital object and its replicas are adaptively placed among different zones, avoiding co-location at zone level. The failure tolerance is then reduced to zone level. In a similar vein, if there is only one zone, a digital object and its replicas are preserved on separate storage nodes; if there is only one storage node, a digital object and its replicas are placed on separate hard disks. In a production deployment, it is strongly recommended to have three or more zones.

After digital objects are ingested into the system, they are organized in logical hierarchy: account, container, and object. An account has a set of containers. A container, similar to a logical file folder, contains many objects. An object, in essence a bit stream, is the smallest storage unit. It can be associated with extensive metadata to further characterize the preserved objects. Access control policy can be customized and enforced. Versioning can be applied to individual digital objects. A suite of consistency services--auditor, replicator, and updater--run constantly in the background in each storage node to check integrity and consistency. If an object is found to be corrupted due to bit-rot or to be missing, the background consistency services work jointly to restore it from two or more other good copies.

Generally speaking, OpenStack Swift has many characteristics making it an ideal bit-level storage candidate for LTDP. An incomplete list of characteristics alongside the LTDP challenges they address are summarized in Table 1 below:

Table 1. Matching the challenges for the bit-level digital storage systems of LTDP with corresponding features of OpenStack Swift.

LTDP Challenges	Comments	Swift Characteristics
Storage scalability	Digital data grows exponentially, leading to ever-increasing demands for storage space.	More hard drives or more storage nodes can be dynamically added, scaling the capacity up.
Object scalability	The number of preserved digital objects multiply, stressing the storage systems, degrading the performance of data access and integrity check.	Swift uses a hash algorithm to map object location to the exact partition of a particular hard disk. Read and write performance scales extremely well. Account, container, object structure allows for billions of objects.
Sustainability	The fund for LTDP is not guaranteed over years to come. Budget cuts are commonplace. It is hard to sustain LTDP	Swift is open-source freeware, requiring only commodity hardware and storage media (HDD), low maintenance, cost-effective and sustainable.
Hardware degeneration	HDD deteriorates/fails over time causing data loss. Refreshing data is a constant need. Servers break down causing disruption of access and services	Swift is designed with redundancy in mind, and thus agnostic of hardware/HDD failure. Refreshing data from an old medium to a new one is simple. No data loss, no access disruption, no service interruption within allowed failure boundary.
Software obsolescence	Operating systems, software enabling the	Swift is compatible with all major operating systems. Swift software is developed in the

LTDP Challenges	Comments	Swift Characteristics
	storage may become obsolete.	popular Python language, and compatible with Amazon S3 API.
Content corruption	Bit-rot or other reasons causing content corruption	Swift can auto detect corrupted files, and recover the content with a good copy, a crucial "self-healing" ability.
Authenticity/Integrity	Maintaining authenticity and integrity is a key requirement for LTDP. It demands constant effort.	Swift has cleverly designed algorithms and data structures, allowing services running in the background to check and enforce authenticity/integrity efficiently and constantly. Access control policy, versioning, and metadata are instrumental to enhance authenticity.
Availability & Reliability	Preserved items need to survive natural disasters, human errors, hardware failures, with high availability and strong reliability.	Swift keeps three or more copies of each object at different locations, even different regions. Such redundancy enables high availability of objects, and reliability of services. Swift can protect accounts from deletion due to mistakes by setting a retention period after the delete command is executed.

Implementation

The configuration of production deployment in the University of Alberta Library organization is illustrated in Table 2 and Table 3 below. The system consists of an authentication node, a proxy node, and six storage nodes. The six storage nodes are in the same region, distributed at three physically separate data centers; and each node is within its own zone. Each storage node has 32GB RAM, 18 hard disks of 4TB each, and a subtotal of 72TB. There are two sockets for each storage node, and eight cores with two threads per core, equivalent to 32 CPUs. The grand total of raw storage space for the storage platform is 432TB. With such an arrangement, the storage system can tolerate failure of two data centers at the same time for read operations; and one data center for write operations. If failure happens at zone level, the tolerance of failure is up to four zones for reading, and up to three zones for writing.

The hardware for the storage nodes is commodity Supermicro machines as shown in Figure 2. There are 24 disk slots in the front of the machine, and 12 additional slots in the back. The storage space can be expanded by filling all of the slots, or replacing the old disks with smaller storage capacity by new ones with larger storage capacity. In addition, the storage node can be further expanded by attaching a SAS3 storage expander with more disk slots. More storage nodes can be purchased and added to the storage system, provided there is a need and budget allows. The 32GB RAM and 32 CPUs are sufficient for the running of background services to check and enforce authenticity, integrity and consistency.

Table 2: Names, roles, types, assigned computing resources for each computer node constituting the private OpenStack Swift storage cloud at UAL.

Name	Role	Type	Memory	CPU	Disks	Storage	Zone	Location
auth	Authentication	Virtual	4GB	2	N/A	N/A	N/A	N/A
proxy	Swift proxy	Virtual	8GB	8	N/A	N/A	N/A	N/A
snode01	Object storage	Hardware	32GB	32	18	72TB	Zone 1	Data centre 1
snode02	Object storage	Hardware	32GB	32	18	72TB	Zone 2	Data centre 1
snode03	Object storage	Hardware	32GB	32	18	72TB	Zone 3	Data centre 2
snode04	Object storage	Hardware	32GB	32	18	72TB	Zone 4	Data centre 2
snode05	Object storage	Hardware	32GB	32	18	72TB	Zone 5	Data centre 3
snode06	Object storage	Hardware	32GB	32	18	72TB	Zone 6	Data centre 3

Table 3: Network configuration settings of and services running on each node. For privacy and security purposes, actual IP addresses are replaced with non-routable ones.

Name	IP Address	Gateway	Services	Firewall ports
auth	192.168.1.10/24	192.168.1.1	httpd, MySQL database server	5000
proxy	192.168.1.11/24	192.168.1.1	proxy service	8080
snode01	192.168.1.12/24	192.168.1.1	account, container, object services	6000,6001,6002
snode02	192.168.1.13/24	192.168.1.1	account, container, object services	6000,6001,6002
snode03	192.168.1.14/24	192.168.1.1	account, container, object services	6000,6001,6002
snode04	192.168.1.15/24	192.168.1.1	account, container, object services	6000,6001,6002
snode05	192.168.1.16/24	192.168.1.1	account, container, object services	6000,6001,6002
snode06	192.168.1.17/24	192.168.1.1	account, container, object services	6000,6001,6002

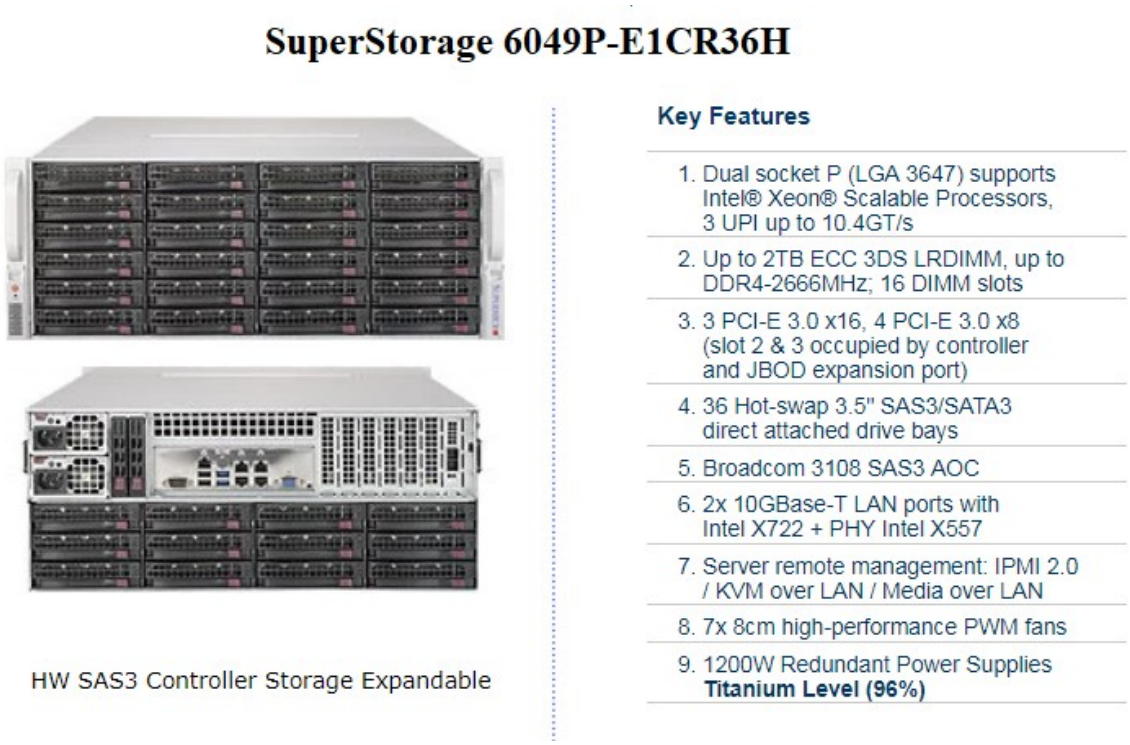

Figure 2: A Supermicro machine with 36 disk slots, 32 CPUs and 32GB RAM.

All nodes were installed with CentOS, a distribution of Linux, which is freely available, and stable with great community support. In fact, OpenStack Swift is compatible with a wide variety of distributions: RedHat, Debian, Ubuntu, openSUSE and SUSE enterprise, relieving concerns over software obsolescence in this regard.

The installation procedures for OpenStack Swift software packages are well documented for the latest release (OpenStack Swift Documentation (OSD).[2] OpenStack releases a new version every six months. Instructions for installation of previous versions are easy to find on the web site as well. Generally speaking, there are three types of installation: proxy nodes, storage nodes, and authentication nodes. To facilitate, streamline and automate the deployment process, we implemented an Ansible playbook for each type of installation. For simplicity of illustration, the Ansible playbook and its associated configuration settings for deploying the Swift proxy node are presented in the code snippet, omitting the Ansible playbooks for the storage and authentication nodes. The purpose of presenting Ansible playbooks is to show a way of deployment with ease and automation. With moderate knowledge of Linux command line environments, deployment of OpenStack Swift is not a daunting but a manageable task.

[2] OpenStack Swift Documentation (OSD). Retrieved from https://docs.openstack.org/swift/latest/

(a) Ansible playbook: sproxy.yml

```
---
 - name: openstack swift proxy node
   hosts: ["proxy"]
   gather_facts: yes
   remote_user: root
   become: no
   vars:
     - release: ussuri
   roles:
    - { role: 'sproxy', tags: sproxy }
```

(b) Settings: vars/main.yml

```
---
packages:
  - python3
  - memcached
  - python3-memcached
  - python3-swiftclient
  - openstack-selinux
  - openstack-swift-proxy
rings:
  - account.builder
  - container.builder
  - object.builder
  - account.ring.gz
  - container.ring.gz
  - object.ring.gz
configs:
  - {name: swift, owner: swift, group: swift, dest: /etc/swift,
     mode: 0660, notify: "openstack"}
  - {name: proxy-server, owner: swift, group: swift, dest: /etc/swift,
     mode: 0660, notify: "openstack"}
  - {name: memcached, owner: root, group: root, dest: /etc/sysconfig,
     mode: 0644, notify: "memcached"}
services: ['memcached', 'openstack-swift-proxy']
firewall_ports: [8080,11211]
```

(c) Ansible role: tasks/main.yml

```
---
- name: install openstack repository
  yum:
    name: "centos-release-openstack-{{release}}"
    state: latest

- name: install swift packages and its dependency
  yum:
    name: "{{packages}}"
    state: latest

- name:copy ring files {account, container, object}.{builder,ring.gz}
  copy: src="{{target_role}}/{{item}}" dest="/etc/swift/{{item}}"
        group=swift mode=0644  owner=swift
  with_items: "{{rings}}"

- name: configure swift-proxy and memcached services
  template: src="{{release}}/{{target_role}}/{{item.name}}.conf.j2"
            dest="{{item.dest}}/{{item.name}}.conf" group="{{item.group}}"
            mode="{{item.mode}}" owner="{{item.owner}}"
  with_items: "{{configs}}"
  notify: restart {{item.notify}}

- name: enable services
  service: name={{item}} enabled=true state=restarted
  with_items: "{{services}}"

- name: firewall
  firewalld: port={{item}}/tcp zone=public permanent=true state=enabled
             immediate=yes
  with_items: "{{firewall_ports}}"


- name: force yum update
  yum: name=* state=latest
```

Figure 3. Ansible playbooks to automate the deployment of the proxy node by sproxy.yml in (a) as the driving code. (b) is the configuration file containing settings for the installation. (c) contains the actual installation steps.

With the help of the respective Ansible playbooks for storage nodes, proxy and authentication nodes, the deployment is reduced to running the following commands with right configuration settings defined:

```
ansible-playbook –ask-vault-pass auth.yml
ansible-playbook –ask-vault-pass sproxy.yml
ansible-playbook –ask-vault-pass storage.yml
```

A Case Study

Sample Digital Collections

As the second largest academic library in Canada, UAL has diverse digital collections, both born-digital and digitized, of intellectual and cultural value that warrant long-term preservation for future generations. For instance, ERA, an acronym for Education and Research Archive, is an open-access institutional repository (IR) of UAL.[3] It encompasses theses, dissertations, intellectual output of researchers, and teaching materials with enduring value. DIGITIZATION collection is the product of UAL's own digitization endeavour to preserve the history of the Canadian West and the culture of the Canadian prairies. Part of this diverse and rich collection consists of "approximately 7,500 digitized books, over 66,000 news issues (4.8 million articles!), 16,000 postcards, and 1,000 maps.".[4] In addition, UAL collaborates with organizations across

[3] ERA Education and research archive. Retrieved from: https://era.library.ualberta.ca/

[4] *Peel's Prairie Provinces.* Retrieved from: http://peel.library.ualberta.ca/

Canada. CWRC, an abbreviation for the Canadian Writing Research Collaboratory, is an online repository of digital scholarly resources, digitized texts, images, audios, videos, and metadata. These digital materials are nation-wide cultural knowledge and literary heritage, which stand "to be lost entirely in the long term, creating what has been called a second dark age."[5] CIHM, which stands for Canadian Institute of Historic Microreproduction, was established by Canadian Council in 1978 with a mandate to preserve "works printed or published in Canada, about Canada, or written by Canadians from as early as the 17th century and as recent as the 1920s." [6]The collection was originally preserved on microfiche, and was digitized later on and preserved at many research libraries across Canada including UAL.

These represent some digital collections preserved by UAL which are tabulated dataset by dataset in Table 4 with the number of digital objects; the minimum and maximum size of files; the average, the median and the total size. The variation of the size of the digital objects is dramatic, ranging from a couple of hundred bytes to approximately 5GB. The size distribution is also presented in Figure 4. CWRC is dominated by files smaller than 50KB, approximately 87%. In contrast, close to 47% of DIGITIZATION collections are files bigger than 50MB. ERA and CIHM are relatively evenly distributed with a smaller spike on either side of the size spectrum. These testify to OpenStack Swift's capability of handling heterogeneous digital objects with size ranging from hundreds of bytes to multiple gigabytes.

Table 4: A representative digital datasets preserved by the on-premises OpenStack Swift cloud storage system at UAL.

Dataset	Num. of Objects	Min. (B)	Max. (B)	Average (B)	Median(B)	Total
CIHM	323,073	122	2,058,002,525	24,415,959	1,674,129	7.88TB
CWRC	405,635	867	5,190,428,915	2,978,182	7,805	1.21TB
DIGITIZATION	19,975	10,240	5,115,289,600	299,920,032	42,598,400	5.91TB
ERA	60,710	35,840	1,881,790,976	8,751,864	3,447,808	531GB

[5] *The Canadian Writing Research Collaboratory*. Retrieved from: https://cwrc.ca/

[6] Canadian Institute for Historic Microreproduction/Early Canadiana. Retrieved from https://www.uvic.ca/library/locations/home/microforms/CIHMhandout.pdf

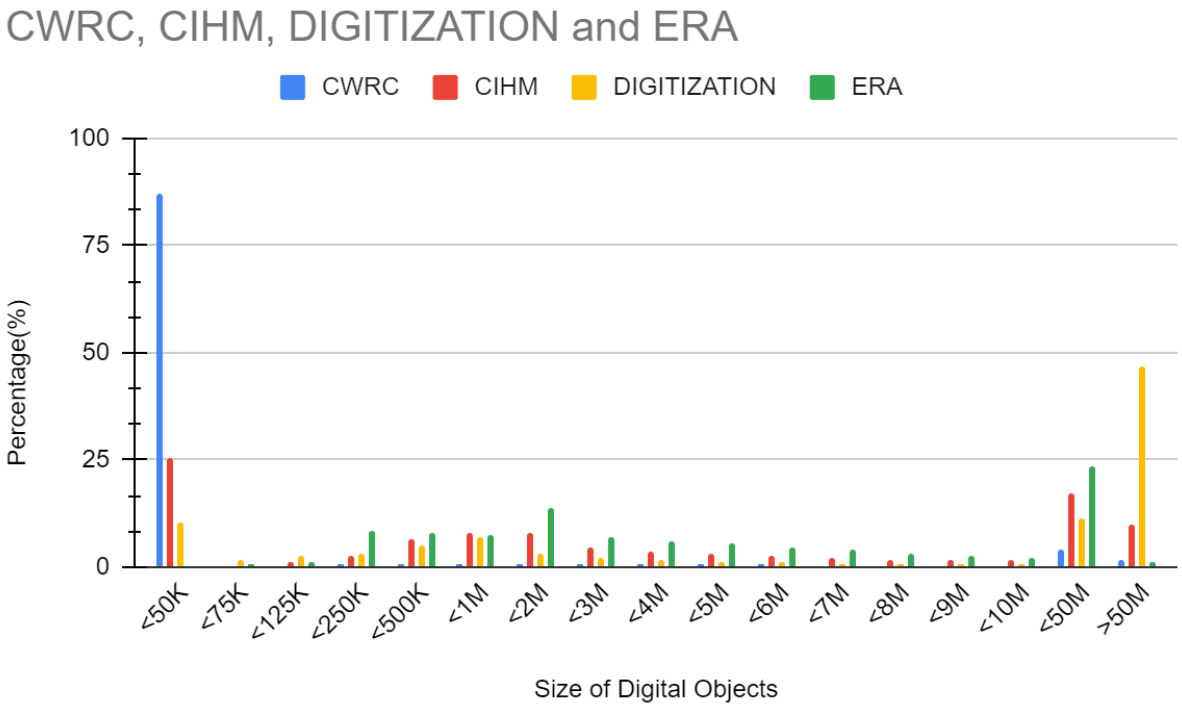

Figure 4: the file size distribution of different collections. CWRC contains dominantly files smaller than 50KB; DIGITIZATION mainly comprises larger files (>50M); ERA and CIHM are relatively evenly distributed.

Uploading and Retrieving Performance

OpenStack Swift is designed to be massively scalable, capable of storing billions of digital objects and hundreds of petabytes of data with virtually no disturbances. In order to shed light on the uploading and downloading performance from containers with a large difference between the numbers of objects, the CWRC container (with 405,635 objects) and an empty Demo container were chosen. The six object files used for the test have different sizes: 250MB, 500MB, 750MB, 1GB, 2GB and 3GB. Each object was uploaded to and downloaded from both CWRC and Demo containers three times, and the average time taken to finish the uploading or downloading process is used for comparison. Figure 5 demonstrates the uploading times; Figure 6 shows the downloading times. The results show that both downloading and uploading times are virtually independent of the size of a container, taking about the same amount of time to upload or download a file with the same size to or from different containers. With the increase of file size, the time taken to upload or download grows near linearly.

In the scientific realm, Toor et al. (2012) conducted a similar scalability and performance study of OpenStack Swift using data generated from the experiments carried out on the Large Hadron Collider (LHC) of the European Organization for Nuclear Research (CERN). These experiments normally generate about 15 PB data per year which is massive and demands enormous scalability for both capacity and I/O performance. Two of Toor et al.'s goals are to study performance of the storage solution when users extract data with different block sizes from a large dataset; performance of the storage solution when there are a large number of objects in a single, very large container. They uploaded and downloaded data in different block sizes such as 4K, 64KB and 1MB. Our experiments differ from theirs by treating a single digital object as a

whole for uploading and downloading; what is varying is the size of the individual file instead of data block size. They used a single large container for performance testing; we used a large and an empty container to compare. In addition, Toor et al. used a 10GB/s network which is 10 times faster than ours. Thus it is not suitable to compare results by numbers. Despite these differences, the experimental results from both parties substantiate the allegation of the superb scalability of OpenStack Swift in dealing with colossal datasets.

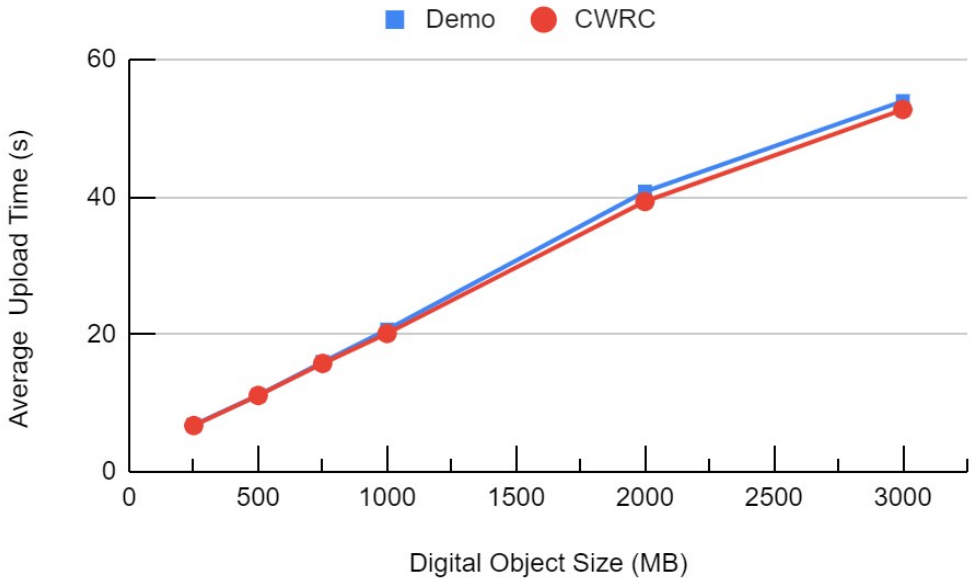

Figure 5. Comparison of uploading times for objects with varying sizes from two vastly different containers.

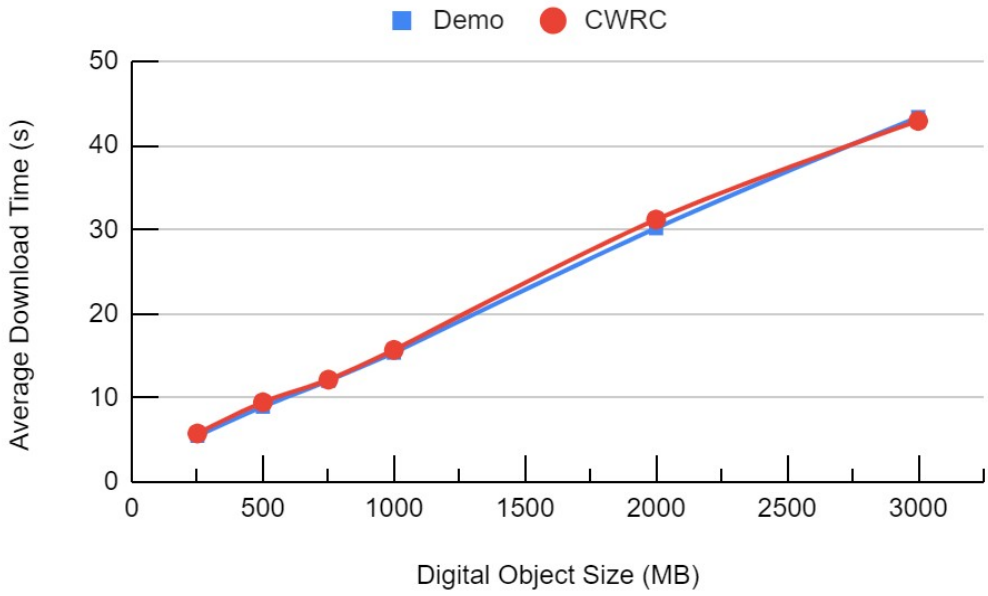

Figure 6: Comparison of downloading times for objects with varying sizes from two vastly different containers.

Maintenance: Software Upgrading and Hardware Deterioration/Failures

Generally speaking, the effort to maintain the OpenStack Swift cluster at UAL is relatively low. The maintenance falls into two broad categories: software upgrades and hardware maintenance.

Software wise, OpenStack Swift has a six month release cycle. A version two or three release cycles older than the most recent stable version is usually not maintained, receiving no updates or security fixes. There are two major implications of running an old version of OpenStack Swift. The first one is the potential security risk. The other is the difficulty of upgrading to the most recent stable version if the current running version is too old. It is possible that a database schema has been modified, making direct migration very hard--if not impossible-- from a version too old and not supported to the most recent version. Thus, it is always a good practice to upgrade the cluster following the official release cycle. To upgrade timely also aligns with the spirit of LTDP to avoid software obsolescence. To reduce upgrading effort and to enhance reliability and consistency, we developed scripts in Ansible to automate the upgrading process.

A second type of software upgrading is the new kernels of the Operating Systems. This requires an ordered reboot of each component node in the system to avoid service disruption. The effort involved is minimal. Yet a more time consuming software upgrade is to upgrade the Operating Systems from one major release to the next one. For example, the OpenStack Swift cluster at UAL was originally deployed to CentOS 6 in 2013. It was upgraded from CentOS 6 to CentOS 7 in 2018. The installation of new Operating Systems on the storage servers requires experiments on test environments prior to applying to the production cluster. It requires both knowledge and careful planning to avoid mistakes that can cause data loss or inaccessibility of data. The effort is indeed moderate.

Hardware wise, HDDs--the storage media of OpenStack Swift--are susceptible to deterioration. Detection of pre-failure signs is conducive to proactive response. There are 108 disks for the UAL OpenStack Swift system, 18 disks per storage node. A cron job in bash script checks each disk for uncorrectable (or bad) sectors on a daily basis. A disk with some bad sectors is still operable because these bad sectors are marked as bad and abandoned by the file systems. No future read or write will be performed on these bad sectors. However, when the number of the bad sectors becomes alarmingly high, it is a forewarning of disk failure and time to replace the failing hard disk. A heatmap is used to visually show which disk on which storage node has bad sectors in Figure 7. From Figure 7, it is shown that the node snode03 has a disk /dev/sdm with one bad sector. The node snode06 has two disks with bad sectors: 296 for the disk /dev/sdk; and 12 for the disk /dev/sdo. Based on our empirical experience, when bad sectors are under 100, the hard disks can still run for quite some time. There is no urgency to replace it right away. If the number of bad sectors increases fast over time, it is a sign of dying. It is better to replace the failing hard disk. Indeed, the disk with 296 bad sectors shown in Figure 7 died in a couple of days.

Instead of using uncorrectable sectors as a litmus test of the health status of HDDs, Kamarthi et al. (2009) used two more parameters: hardware ECC recovery and read/write rate. They developed an artificial neural network to assess the health status and remaining useful lifetime of HDDs with a claimed 88% accuracy. Kamarthi et al.'s approach is more complex than ours, but it corroborates our approach to monitor the health status of HDDs. Considering its good but not great 88% accuracy and the fact that we don't have to wait till the last moment

for a dying HDD to be replaced, it is not necessary to adopt their approach.

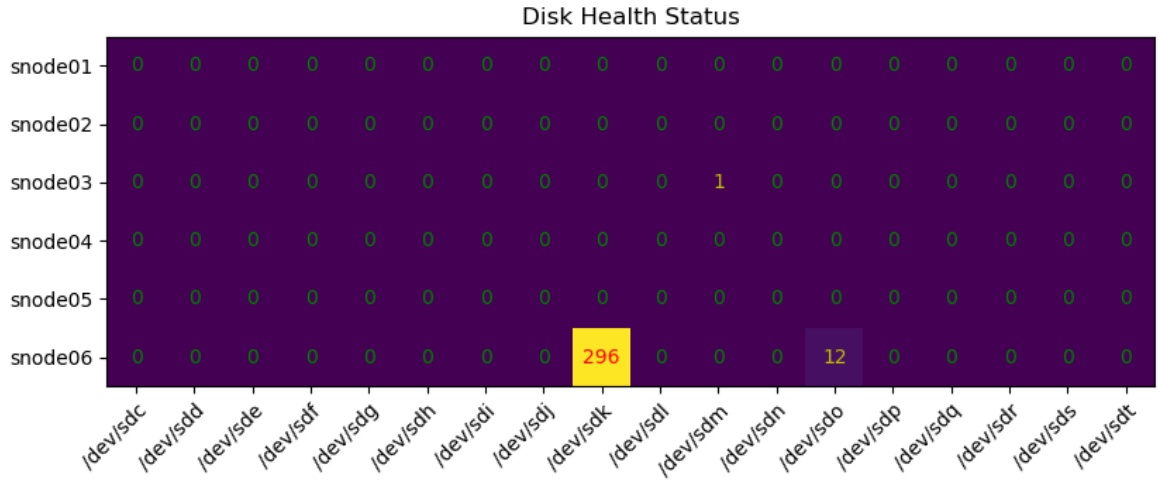

Figure 7: The health status of hard disks. The x-axis is for disk names, and the y-axis is for node names.

It is normal for hard disks to die after running for some time. OpenStack Swift withstands failure of hard disks without disruption of services or loss of data. The failure of hard disks can be monitored and reported by running the built-in health status check command. The failed hard disks can be replaced with new ones formatted in XFS format and mounted properly with correct ownership. The digital objects originally stored on the failed hard disks are automatically replicated back to the new ones from the corresponding copies on other good disks.

Refreshing is a necessary proactive strategy that is vital for the success of LDTP, because storage media becomes unavoidably outdated or degraded, which might lead to bit-rot or alteration of digital objects. With OpenStack Swift, refreshing data from old storage media to new ones is as simple as replacing a failed hard disk. If bit-rot or unintentional changes of digital objects do happen, the background auditing processes are designed to detect the abnormalities and respond with corresponding services to bring the impacted digital objects into a consistent state. These failure scenarios have been tested and verified through the OpenStack Swift system at UAL.

Economics: OpenStack Swift vs Azure cloud

Budget plays an important role in the sustainability of LTDP. It is instrumental in knowing how cost-effective it is to choose OpenStack Swift as the bit-level storage system for LTDP over the commercial clouds such as Azure. Microsoft Azure is chosen for a number of reasons. First, Microsoft Azure provides a tool named Total Cost of Ownership (TCO) for prospective customers to compare costs between on-premises and Azure cloud, for a lift-and-shift migration. Second, other major cloud service providers do not have a tool that is comparable to Microsoft Azure's TCO. Third, Rosenthal (2014) and Breeding (2012) separately conducted a cost comparison between local storage and AWS cloud storage; it is beneficial to conduct a cost comparison different from AWS to widen our view regarding cloud storage cost.

With the TCO calculator, we entered the actual specifications of the currently running OpenStack Swift in the data center of our organization. These include:

- Six identical physical servers for storage each with 32GB RAM, 2 processors, and 8 cores per processor, 2 threads per core, equivalent to 32 CPUs.
- Two virtual machines: 16GB RAM and 8 CPUs for Swift Proxy; and 4GB RAM and 2 CPUs for authentication.
- Raw storage space in HDDs (used and unused): 432TB
- Monthly outbound network usage: 2TB
- Open-source software: Operating Systems; virtualization; application software

On the Azure side, local redundant storage (LRS) rather than geographic redundant storage (GRS) is assumed to reflect the reality of the on-premises configuration. The TCO tool assumes that neither backup nor disaster recovery is configured. The sole purpose is to simulate a lift-and-shift migration from on-premises to the cloud, taking advantage of the cloud's Infrastructure as a Service (IaaS) feature.

The cost is broken down into five categories: Compute; Data Center; Networking; Storage; and IT Labor. The compute cost is further decomposed into hardware, software, electricity and virtualization. The cumulative costs estimated by TCO over a five year period for both on-premises and the corresponding Azure cloud are presented in column 2 and 3 of Table 5. For comparison, our actual on-premises costs based on purchase orders are also presented in column 4 of the table.

For the computing cost, our purchase order shows its hardware part is C$39,330.36, which is the cost for the physical servers. Considering the maintenance of these physical servers, usually 15% of the original purchase price is added as maintenance cost. Therefore, the final hardware cost over five years is C$45,229.91. For the electricity part, the power supply is 1200W which is the maximum allowable. In reality, the power drawn from the power supply is a lot smaller than the maximum. As a conservative estimate, each physical server is assumed to run at 1000W 24 hours a day, 7 days a week. The resulting total electricity cost in five years is C$26,280, assuming C$0.10/kWh which is slightly above the market average rate. Open source operating systems and application software are used. Therefore software cost is reasonably computed as 0, so for virtualization technology. Therefore, the conservative estimate produces a total cost of C$71,509.91 for the computing part, which is 63% of the Azure cloud cost, and 23% of the TCO estimate for the on-premises option. It is clear that TCO over-estimates the on-premises cost by a lot; and Azure cloud computing cost is more expensive than the actual on-premises option.

For the storage cost, the actual purchase price for all the HDDs is C$26,772. The HDDs are more susceptible to failure, a 50% maintenance cost for five years is assumed, which is C$13,386. Therefore, the total storage cost is C$40,158. The Azure cloud storage cost in five years is shown to be C$764,411.90 which is overwhelmingly more expensive than the actual cost. The TCO estimate for the on-premises storage is C$195,349.68, which is 25% of the Azure storage cost. However, it is still a big overestimate in comparison to the actual cost. The high cost of Azure storage for large scale datasets agrees well with the findings of both Rosenthal (2010) and Breeding (2012).

For the networking cost, TCO assumes that firewall hardware, switches, routers and other networking related components are dedicated to the OpenStack Swift. In reality, these hardware components are shared by many other applications of the same organization. In the UAL case, many of these components are provided to UAL for use for free. Therefore, we assume the networking cost is the same as the TCO estimate for the Azure networking cost.

For IT labor cost, TCO underestimate the cost. TCO underestimates the IT labor for the on-premises option. TCO's underestimate of IT labor can be attributed to its assumption of a

low hourly salary rate and a low number of working hours. Hourly salary rate for the same IT job can vary a lot from region to region, or from nation to nation. Thus, the estimate can only be used as a rough guidance. In reality, the system administrator spends about 15% of his total working hours to support the on-premises OpenStack Swift system. Based on the system administrator's five-year salary, the IT labor cost to support the on-premises OpenStack Swift system is about C$70,500.00 over five years.

The most debatable and most ambiguous part of cost is for the data center. TCO estimates data center cost over five years to be C$874,406.70, among which the rack mounting and installation cost about $835,716.10, accounting for 96% or so. This dominance shows that the cost of other parts of a data center is marginal. It is important to note that UAL uses the data centers belonging to the University; UAL is not charged for electricity, server racks, and cooling. Thus, in column 4 of Table 5, the corresponding data center cost is not presented. The authors recognize that not every organization like UAL can be waived from paying costs related to running a data center. Therefore, this part has no meaning for these organizations.

It is reasonable to conclude that 1) TCO tends to overestimate the computing and storage costs for the on-premises options; 2) Azure computing and storage costs are more expensive than the on-premises options; 3) Azure storage cost is costly and dwarfs other costs. In addition, for small organizations that cannot afford or justify the cost of running data centers, Azure cloud is potentially more cost-effective, depending on how much the data center costs. However, for large organizations that can share data center cost, or even better the data center cost is covered by its parent organizations, on-premises is still more cost-effective, especially when large scale datasets are involved.

Category	On-Premise (TCO)(C$)	Azure (TCO)(C$)	On-Premise (Actual)(C$)
Compute	306,032.90	113,621.30	71,509.91
Hardware	265,000.96	0.00	39,330.36
Software	0.0	0.00	0.00
Electricity	36,958.46	0.00	26,280
Virtualization	4,073.50	0.00	0.00
Data Center	874,406.70	0.00	N/A
Networking	68,752.67	1,536.00	1,536.00
Storage	195,349.68	764,411.90	40,158.00
IT Labor	22,550.53	19,609.11	70,500.00
Total	1,467,092.40	899,194.88	183,703.91

Table 5: Cost comparisons between the actual on-premises Swift storage system and its corresponding lift-and-shift Azure cloud migrated system. Considering maintenance costs, the resulting computing cost is 15% more than the hardware purchase price (C$39,330.36). Similarly, the storage cost is 50% more than the original purchase price (C$26,772).

Conclusion and Discussion

In this paper, the distinctive features of OpenStack Swift are elaborated. Its unlimited scalability of storage space, its linear performance with regard to dramatically increasing numbers of objects, its effectiveness and efficiency in enforcing authenticity, integrity and consistency of preserved digital objects, its highly distributed and interconnected nature and configurable

degree of redundancy, its high tolerance of hardware failure, and what is more, its self-healing ability to repair corrupted objects, make it an ideal bit-level storage system for LTDP. In addition, OpenStack Swift's compatibility with commodity hardware and consumer grade storage media, together with its free availability and low maintenance, makes it a cost-effective solution in the face of not uncommon budgetary challenges.

The contribution of this paper is to fill the void in the LTDP community with a systematic introduction of OpenStack Swift, supplemented with both the implementation of the solution and operational results in production environments. It offers a testament of the feasibility and practicability of using OpenStack Swift as a large scale storage system for LTDP. Indeed, the adoption of OpenStack Swift as a viable solution for LTDP, even though not well documented, has gained traction over the past years. The private cloud storage at UAL, the community cloud storage at Scholars Portal, and Japan's NTT's 7PB private cloud, all built on top of OpenStack Swift, offer convincing arguments for the adoption of OpenStack Swift.

However, OpenStack Swift is not proposed as the ultimate solution for bit-level digital preservation. It is universally accepted by the digital preservation community that more than one type of storage media and more than one storage technology are desired for LTDP; the storage media of OpenStack Swift are dominantly hard disk drives; and thus OpenStack Swift is proposed as a competitive alternative to diversify the means for LTDP so that different means can complement one another. One drawback of OpenStack Swift as a bit-level digital preservation system is its eventual consistency. Specifically, if a replica of a digital object has been altered for whatever reason, the change is not detected right away, and the recovery doesn't happen immediately. There is a time delay to bring the altered object back to a consistent state. The length of the delay depends on the performance of the auditing, updating and replicating processes, which in turn depend on the CPU power and the efficiency of algorithms underlying these processes. It is also worth noting that OpenStack Swift alone can't meet other imperative requirements for a full-fledged digital preservation system that conforms to the NDSA levels of digital preservation, for example, file format check and migration (Phillips et al. 2013). It needs to be integrated with other software packages to fulfil these requirements.

In our implementation of the private OpenStack Swift cloud at UAL, collaboration with external organizations was contemplated but not materialized. It's advantageous to leverage OpenStack Swift's support for storage across geographic regions, capitalizing on pooled resources and expertise, increasing tolerance of failure, and cutting cost at the same time. The prospect of inter-organizational collaboration using OpenStack Swift as a vehicle is promising, but the network latency between two organizations which are geographically separated is a barrier to overcome. It is a subject of interest for future research.

Additionally, commercial cloud storage services from Google, Microsoft and Amazon have become readily available. There are contradicting views about the cost-effectiveness of their application as a storage system for LTDP. A preliminary cost comparison between the on-premises OpenStack Swift system at UAL and the commercial Azure cloud demonstrates the cost competitiveness of the former. The study finds that the cost of Azure computing is high, and the storage cost of Azure for large datasets is alarmingly high, which is in good agreement with the findings of other researchers (Rosenthal, 2010; Breeding, 2012). From an economic perspective, if data center cost is not a concern for an organization, on-premises deployment of OpenStack Swift with large storage capacity for LTDP makes more sense. However, a systematic and comprehensive comparative study of cost-effectiveness--one of the principal driving forces for the adoption of commercial cloud storage for LTDP--between the private OpenStack Swift and these public cloud services is lacking, and thus a natural and necessary topic for further research in the near term.

Acknowledgements

The Authors would like to thank Dr. Peter Binkley for his original proposal of adopting OpenStack Swift as a bit-level storage system for LTDP, and for his many insightful suggestions for improvement based on his deep knowledge in the field of LTDP. The authors are equally grateful to John Huck, metadata librarian of UAL, for his critical review and constructive advice. The authors are also deeply indebted to Neil MacGregor for his technical advice and assistance.

References

Askey, D. & Ruest, N. (2016). Ontario library research cloud: From idea to infrastructure in three easy years. Retrieved from: http://hdl.handle.net.login.ezproxy.library.ualberta.ca/11375/19543.

Arnold, J. (2014). *OpenStack Swift: Using, administering, and developing for Swift Object Storage*. O'Reilly Media, Inc.

Breeding, M. (2012). Cloud computing for libraries. ALA TechSource, an imprint of the American Library Association.

Digital Preservation Coalition. (2015). Digital preservation handbook (2nd ed.). Retrieved from https://www.dpconline.org/handbook

Durand, G. (2020). Dataverse's Approach to Technical Community Engagement. Septentrio Conference Series, 2, Article 2. https://doi.org/10.7557/5.5424

Giaretta, D. (2011). *Advanced digital preservation. [electronic resource]*. Springer. https://doi.org/10.1007/978-3-642-16809-3_3

Goodchild, M. & Hurley, G. (2019). Integrating Dataverse and Archivematica for Research Data Preservation. In IPRESS 2019. Retrieved from: https://www.ipres2019.org/static/pdf/iPres2019_paper_147.pdf

Hedstrom, M. (1998). Digital preservation: A time bomb for digital libraries, *Computer and Humanities* **31**: 189-202.

Haughton, B. (2016). Preservation challenges in the digital age. *D-Lib Magazine.* **22**(7), pp. 1-6. doi:10.1045/july2016-houghton

Kamarthi, S., Zeid, A. & Bagul, Y. (2009). Assessment of current health of hard disk drives. *2009 IEEE International Conference on Automation Science and Engineering*. pp. 246-249, doi: 10.1109/COASE.2009.5234105.

Kimpton, M., & Payette, S. (2010). Using Cloud Infrastructure as Part of a Digital Preservation Strategy with DuraCloud. *EDUCAUSE Quarterly,* 33(2), 13.

Kapadia, A., Rajana, K., Varma, S. (2014). *Implementing cloud storage with OpenStack Swift*. Birmingham, UK : Packt Publishing.

Lu, D., & Pan, Y. (2010). *Digital preservation for heritages: technologies and applications.* Springer.

Myntti, D. & Zoom, J. (2019). *Digital preservation in libraries: Preparing for a sustainable digital future.* Chicago : ALA Editions, an imprint of the American Library Association

Phillips, M., Bailey, J., Goethals, A., & Owens, T. (2013). *The NDSA levels of digital preservation: Explanation and uses.* In *Archiving conference* (Vol. 2013, No. 1, pp. 216-222). Society for Imaging Science and Technology.

Rieger, O.Y. (2018). The state of digital preservation in 2018: A snapshot of challenges and gaps. *Ithaka S+R.* https://doi.org/10.18665/sr.310626

Rosenthal, D.S.H. (2010). Bit preservation: A solved problem?" *The International Journal of Digital Curation*. **5** (1). Retrieved from: http://www.ijdc.net/index.php/ijdc/article/view/151/224

Rosenthal, D.S.H., Robertsoni,T., Lipkisii, T., Reichi, V., & Morabitoi, S. (2005). Requirements for digital preservation systems: A bottom-up approach. *D-Lib Magazine*, **11**(11). Retrieved from http://www.dlib.org/dlib/november05/rosenthal/11rosenthal.html

Rosenthal, D.S.H. & Vargas, D.L. (2013). Distributed digital preservation in the cloud, *8th International Digital Curation Conference*. Retrieved from https://web.stanford.edu/group/lockss/resources/2018-01_Distributed_Digital_Preservation_in_the_Cloud.pdf

Rosenthal, D.S.H. (2014). Costs: why do we care? *DSHR's Blog*. Retrieved from: https://blog.dshr.org/2014/11/talk-costs-why-do-we-care.html

Toor, S., Holmgren, S., Töebbicke, R., & Resines, M. Z. (2012). Investigating an open source cloud storage infrastructure for CERN-Specific data analysis. Proceedings – 2012 IEEE 7th International Conference on Networking, Architecture and Storage, NAS pp. 84–88. doi:10.1109/NAS.2012.14.

Wilcox, D. & Weinraub, E., (2017). Supporting digital preservation and access with Fedora. *IFLA World Library Information Congress Conference*. Paper presented at: IFLA WLIC 2017 – Wrocław, Poland – Libraries. Solidarity. Society. in Session 150 - Preservation and Conservation, Acquisition and Collection Development. Retrieved from: https://library.ifla.org/id/eprint/1758