How swift is your Swift?

Ning Zhang, OpenStack Engineer at Zmanda
Chander Kant, CEO at Zmanda

Outline

- **Build a cost-efficient Swift cluster with expected performance**
  - Background & Problem
  - Solution
  - Experiments
- When something goes wrong in a Swift cluster
  - Two Types of Failures: Hard Drive, Entire Node
  - What is performance degradation before the failures are fixed
  - How soon the data will be back (when all failed nodes are back on-line)?
  - Experiments

zmanda
Open Source Backup

Zmanda

- Leader in Open Source Backup and Cloud Backup
- We got strong interest in integrating our cloud backup products with OpenStack Swift
- Backup to OpenStack Swift
  - Alternative to tape based backups
- Swift Installation and Configuration Services

zmanda
Open Source Backup

Background

- Public Storage Cloud
  - Pros: pay-as-you-go, low upfront cost …
  - Cons: expensive in the long run, performance is not clear …
- Private Storage Cloud (**use case**: backup data to private cloud by Zmanda products)
  - Pros: low TCO in the long run, expected performance, in-house data …
  - **Cons**: high upfront cost, long ramp-up period (prepare and tune HW & SW)
- **Open Problem / Challenge:**
  - How to build a private cloud storage with ….
  - Low upfront cost, expected performance, short ramp-up period

Background

- **Swift** is an open-source object store running on commodity HW
  - High scalability (linear scale-out as needed)
  - High availability (3 copies of data)
  - High durability
- Swift has heterogeneous types of nodes

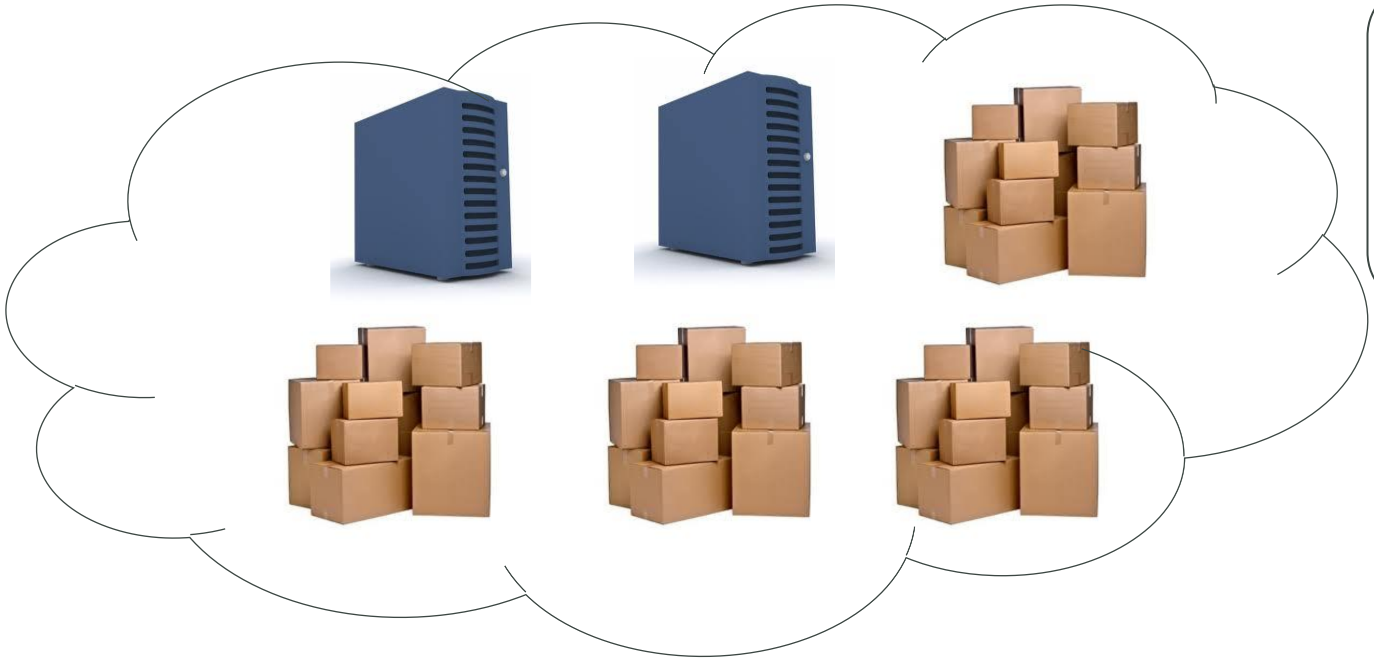
Proxy – Swift's brain (coordinate requests, handle failure…)

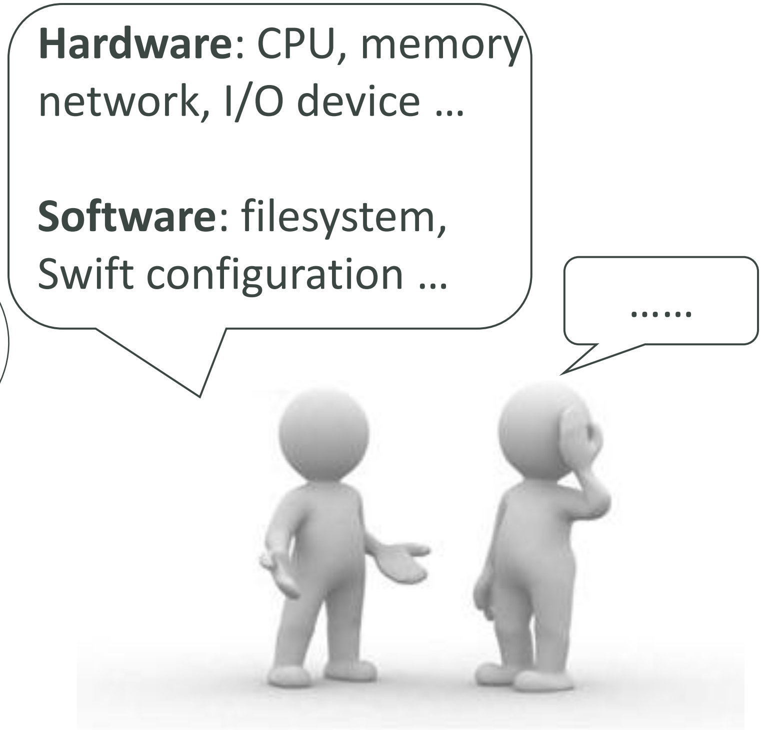
Storage – Swift's warehouse (store objects)

zmanda
Open Source Backup

Problem

- How to provision the proxy and storage nodes in a Swift cluster for <u>expected performance (SLA)</u> while keeping <u>low upfront cost</u>?





zmanda Open Source Backup

Lesson Learnt from Past

CPU, network I/O intensive

High-end CPU, 10 GE networking

Disk I/O intensive

Commodity CPU, 1 GE networking

XFS filesystem

Are they **<u>always true</u>** in all cases? especially for different workloads?

- always pay off to choose 10GE (expensive!) for proxy nodes?
- always sufficient to use commodity CPU for storage nodes?
- always disk I/O intensive on storage nodes?
- how much difference in performance between XFS and other FS?
- ….

zmanda Open Source Backup

Solution

- Solution (similar to "**Divide and Conquer**" strategy)
  - First, solve the problem in a small Swift cluster (e.g. 2 proxy nodes, 5-15 storage nodes)

1: **For each** HW configuration for proxy node
2: **For each** HW configuration for storage node
3: **For each** number of storage node from 5, 10, 15…
4: **For each** SW parameter setting
5: A **small Swift cluster** is made, measure its performance, calculate and save its "**perf/cost**"
6: Recommend the small Swift clusters with high "performance/cost"

Exhaustive search?

Pruning methods make it simple!

  - Then, **scale out** recommended small Swift clusters to large Swift clusters until SLA is met
    - Performance and cost also get scaled (when networking is not a bottleneck)
    - The ratio between proxy and storage nodes (identified in small clusters)
    - As well as HW & SW settings (identified in small clusters) still *hold* in large clusters

Evaluation - Hardware

- Hardware configuration for proxy and storage nodes
  - Amazon EC2 (diverse HW resources, no front cost, virtualized HW -> physical HW)
  - Two hardware choices for proxy node:
    - # 1: Cluster Compute Extra Large Instance (**EC2 Cluster**)
    - # 2: High-CPU Extra Large Instance (**EC2 High-CPU**)
  - Two hardware choices for storage node:
    - # 1: High-CPU
    - # 2: Large Instance (**EC2 Large**)

	Cluster	**High-CPU**	**Large**
CPU speed	33.5 EC2 Compute Units	20 EC2 Compute Units	4 EC2 Compute Units
Memory	23 GB	7GB	7.5GB
Network	**10 GE**	1 GE	1 GE
Pricing (US East)	$ 1.30/h	$ 0.66/h	$ 0.32/h

zmanda
Open Source Backup

Evaluation – Cost & Software

- Upfront Cost
  - EC2 cost ($/hour)
  - EC2 cost≠ physical HW cost, but it is a good implication of physical HW cost
- Software Configuration
  - Filesystem
    - XFS (recommended by RackSpace)
    - Ext4 (popular FS, but not evaluated for Swift)
  - Swift Configuration Files
    - *db_preallocate* (it is suggested to set True for HDD to reduce defragmentation)
  - OS settings
    - disable TIME_WAIT, disable syn cookies …
    - will discuss in our future blog …

Evaluation – Workloads

- 2 Sample Workloads

	Upload (GET 5%, PUT 90%, DEL: 5%)
Small Objects (object size 1KB – 100 KB)	**Example: Online gaming hosting service** the game sessions are periodically saved as small files.
Large Objects (object size 1MB – 10 MB)	**Example: Enterprise Backup** the files are compressed into large trunk to backup. Occasionally, recovery and delete operations are needed.

Object sizes are randomly and uniformly chosen within the pre-defined range
Objects are continuously uploaded to the test Swift clusters

- COSBench – a cloud storage benchmark tool from intel
- Free to define your own workloads in COSBench !

zmanda
Open Source Backup

Evaluation – Upload small objects

- Top-3 recommended hardware for a small Swift cluster

		HW for proxy node	HW for storage node	Throughput/$
Upload Small Objects	**1**	**2 proxy nodes (High-CPU)**	**5 storage nodes (High-CPU)**	151
	2	2 proxy nodes (Cluster)	10 storage nodes (High-CPU)	135
	3	2 proxies nodes (Cluster)	5 storage nodes (High-CPU)	123

- **Storage node** is all based on **High-CPU**
  - CPU are intensively used for handling large # requests. CPU is the key resources.
  - Comparing to **Large** Instance (4 EC2 Compute Units, $0.32/h)
  - **High-CPU** Instance has 20 EC2 Compute Units with $0.66/h (5X more CPU resources, only 2X expensive)

- **Proxy node**
  - Traffic pattern: high throughput, low network bandwidth (e.g. 1250 op/s -> 61MB/s)
  - 1 GE from **High-CPU** Instance will not be the serious bottleneck for this workload.
  - Comparing to High-CPU, **Cluster** has 1.67X CPU resources, but 2X expensive
  - Besides, 5 **High-CPU** based storage nodes can almost saturate 2 **High-CPU** based proxy nodes

Evaluation – Upload large objects

- Top-3 recommended hardware for a small Swift cluster

		HW for proxy node	HW for storage node	Throughput/$
Upload Large Objects	**1**	**2 proxy nodes (Cluster)**	**10 storage nodes (Large)**	5.6
	2	2 proxy nodes (High-CPU)	5 storage nodes (Large)	4.9
	3	2 proxy nodes (Cluster)	5 storage nodes (Large)	4.7

- **Storage node** is all based on **Large**
  - More time is spent on transferring objects to I/O devices. Write request rate is low, CPU is not the key factor.
  - Comparing to **High-CPU** Instance (20 EC2 Compute Units, $0.66/h),
  - **Large** Instance has 4 EC2 Compute Units (sufficient) with $0.32/h (2X cheaper).

- **Proxy node**
  - Traffic pattern: low throughput, high network bandwidth
  - e.g. 32 op/s -> 160 MB/s for incoming and ~500 MB/s for outgoing traffic (write in triplicate!)
  - 1 GE from High-CPU is under-provisioned, 10 GE from Cluster is **paid off** for this workload**.**
  - Need 10 **Large** based storage nodes to keep up with the 2 proxy nodes (10 GE)

Evaluation – Conclusion for HW

- Take-away points for provisioning **HW** for a Swift cluster

	Hardware for proxy node	**Hardware for storage node**
Upload Small Object	1 GE High-end CPU	1 GE High-end CPU
Upload Large Object	10 GE High-end CPU	1 GE Commodity CPU

- Download workloads: see the backup slides
- Contrary to the lessons learnt from the past
  - It does **NOT** always pay off to choose 10 GE (expensive!) for proxy nodes
  - It is **NOT** always sufficient to use commodity CPU for storage nodes
  - Upload is disk I/O intensive (3 copies of data)
  - but download is **NOT** always disk I/O intensive (retrieve one copy of data)

Evaluation – Conclusion for SW

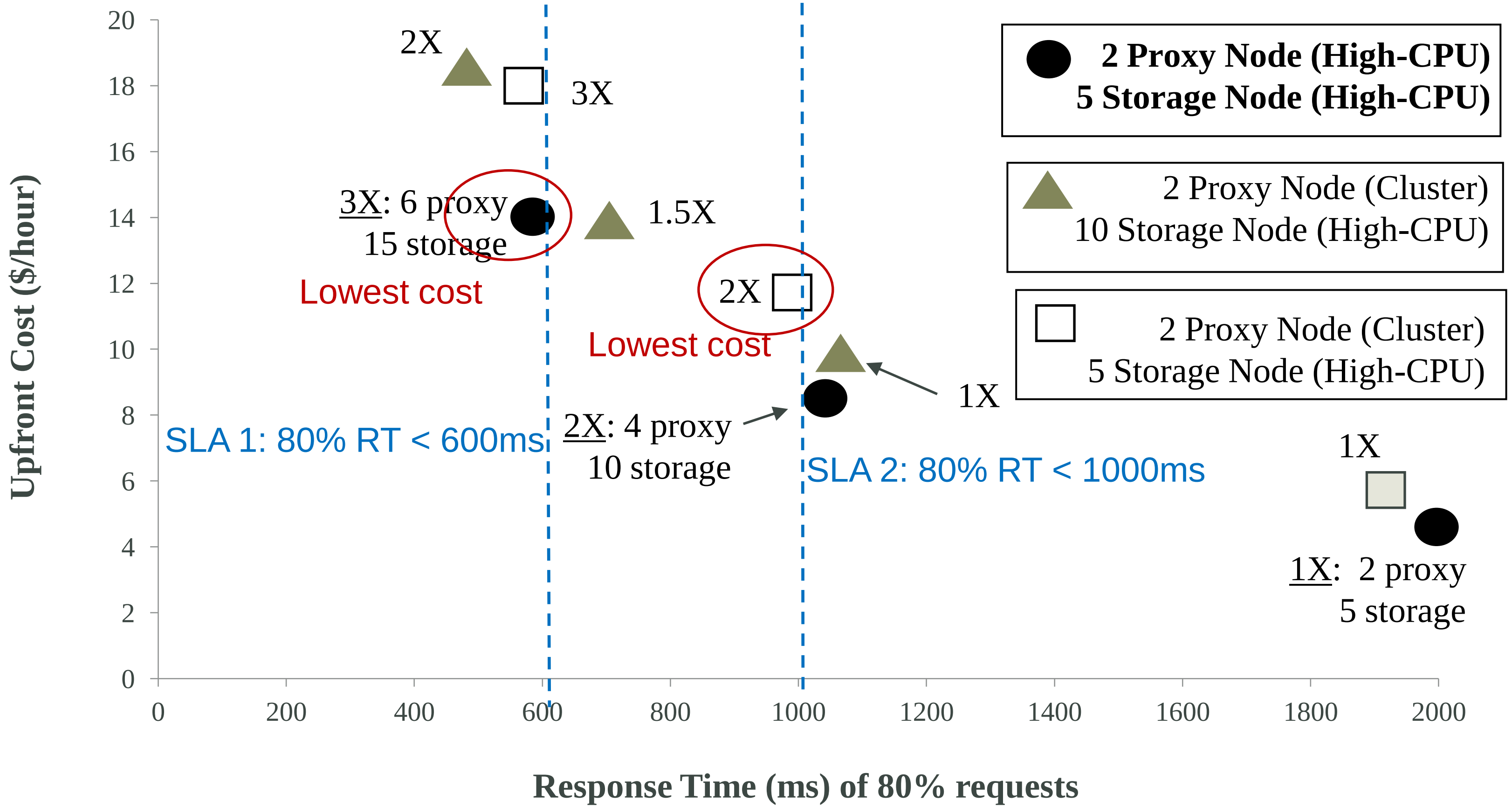
- Take-away points for provisioning **SW** for a Swift cluster

	db_preallocate	XFS vs. Ext4
Upload Small Objects	on	XFS
Upload Large Objects	on / off	XFS / Ext4

- Upload Small Object (more sensitive to software settings)
  - db_preallocation: intensive updates on container DB. Setting it to **on** will gain 10-20% better performance
  - Filesystem: we observe XFS achieves 15-20% extra performance than Ext4

Evaluation – Scale out small cluster

- Workload #1: upload small objects (same workload for exploring HW & SW configurations for small Swift cluster)
- Based on the top-3 recommended small Swift clusters



zmanda
Open Source Backup

Outline

- Build a cost-efficient Swift cluster with expected performance
  - Background & Problem
  - Solution
  - Experiments
- **When something goes wrong in a Swift cluster**
  - Two Types of Failures: Hard Drive, Entire Node
  - What is performance degradation before the failures are fixed
  - How soon the data will be back (when all failed nodes are back on-line)?
  - Experiments

zmanda
Open Source Backup

Why Consider Failures

- **Failure stats** in Google's DC (from Google fellow Jeff Dean's interview at 2008)
  - A cluster of **1,800 servers** in its first year……
  - Totally, **1,000 servers failed**, thousands of HDDs failed
  - 1 power distribution unit failed, bringing down 500 – 1,000 machines for 6 hours
  - 20 racks failed, each time causing 40 – 80 machines to vanish from network

- Failures in Swift
  - Given a 5-zone setup, Swift can tolerate **at most** 2 zones failed (data will not be lost)
  - But, performance will degrade to some extent before the failed zones are fixed.
  - If Swift operators want to ensure certain performance level
  - They need to benchmark the performance of their Swift clusters upfront

zmanda
Open Source Backup

How Complex to Consider Failure

- (1) Possible failure at one node
  - Disk
  - Swift process (rsync is still working)
  - Entire node
- (2) Which type of node failed
  - Proxy
  - Storage
- (3) How many nodes failed at same time
- Combining above three considerations, the total space of all failure scenarios is huge
  - practical to prioritize those failure scenarios
  - E.g. the worst or more common scenarios are considered first

zmanda
Open Source Backup

Evaluation - Setup

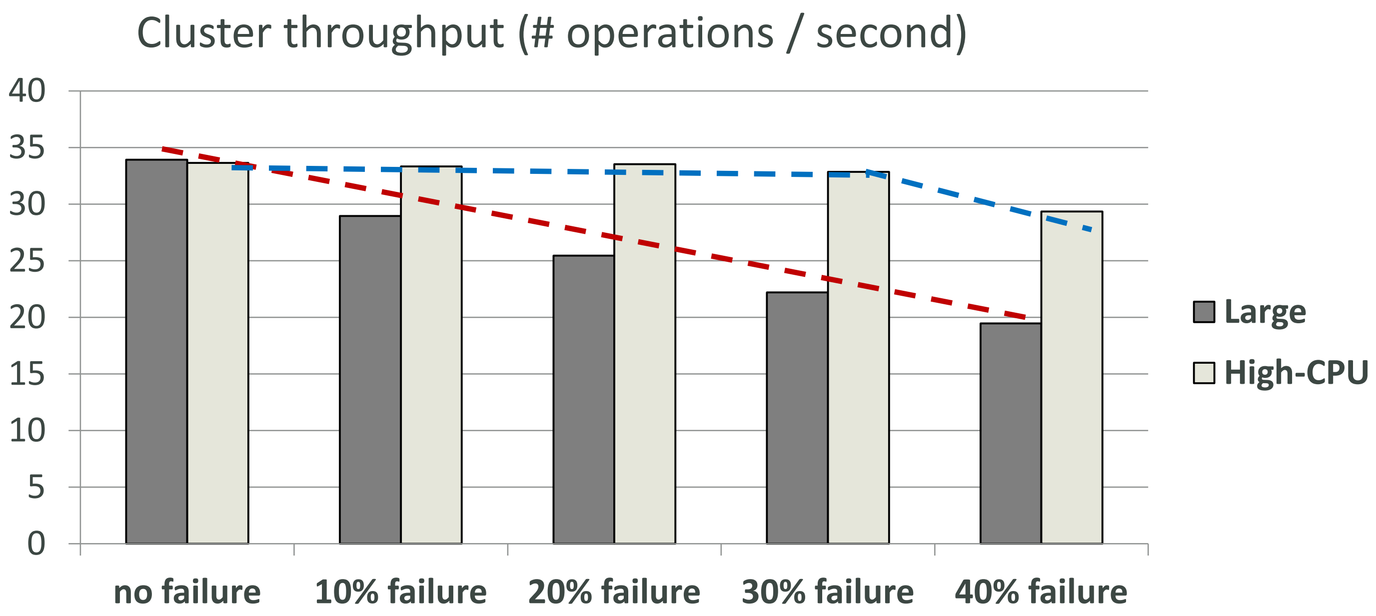
- Focus on performance (not data availability)
- Measure performance degradation comparing to “no failure” case, before failed nodes back on-line
- Workload: Backup workload (uploading large objects is the major operation)
- Swift cluster: 2 proxy nodes (Cluster: Xeon CPU, 10 GE), 10 storage nodes
- **Two common** failure scenarios: (1) entire storage node failure (2) HDD failure in storage node
- **(1) Entire storage node failure**
  - **10%, 20%, 30% and 40%** storage nodes failed in a cluster (E.g. partial power outage)
  - Different HW resources are provisioned for storage node
    - EC2 Large for storage node (cost-efficient, high performance/cost)
    - EC2 High-CPU for storage node (costly, over-provisioned for CPU resources)

zmanda
Open Source Backup

Evaluation - Setup

- **(2) HDD failure in storage node** (EC2 Large for storage node)
  - Each storage node attaches 8 HDDs
  - Intentionally *umount* some HDDs during the execution.
  - Storage node is still accessible
  - **10%, 20%, 30% and 40%** of HDDs failed in a cluster
  - Compare two failure distributions:
    - **Uniform** HDD failure (failed HDDs uniformly distributed over all storage nodes)
    - **Skewed** HDD failure (some storage nodes get much more # HDDs failed than other nodes)

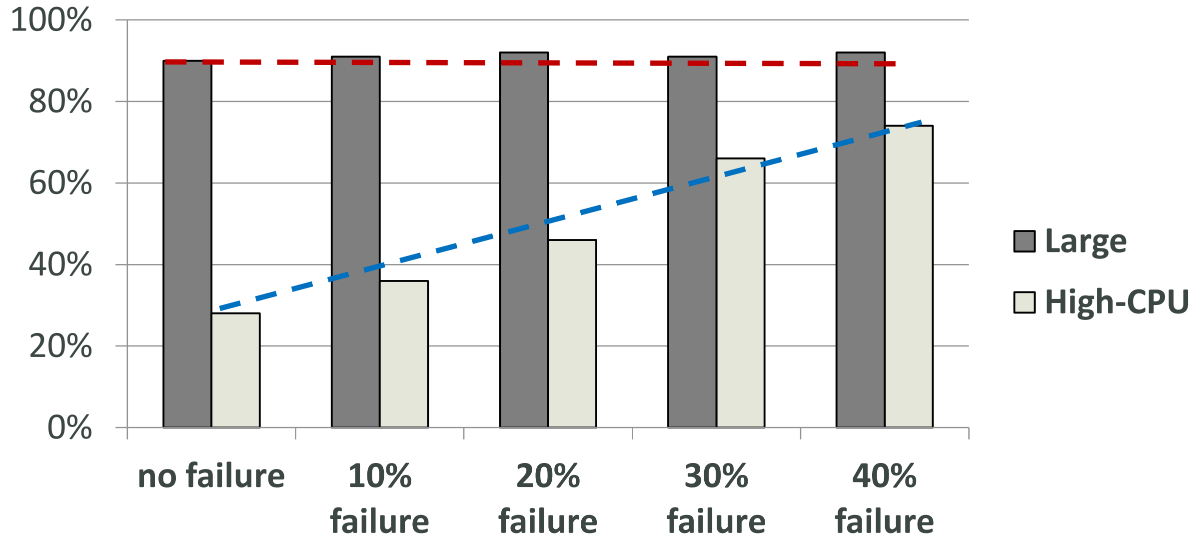
Evaluation – Entire Node Failure



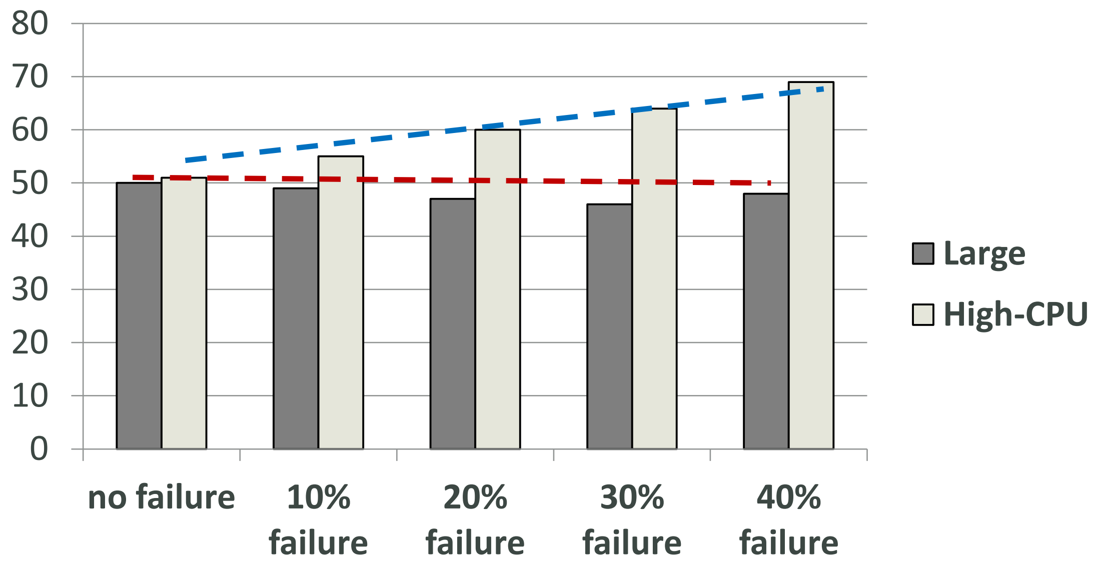
- **Storage node based on Large Instance**
  - Throughput decreases as more storage failed
- **Storage node based on High-CPU Instance**
  - Throughput decreases only when 40% nodes fail

Evaluation – Entire Node Failure

CPU usage in unaffected storage node

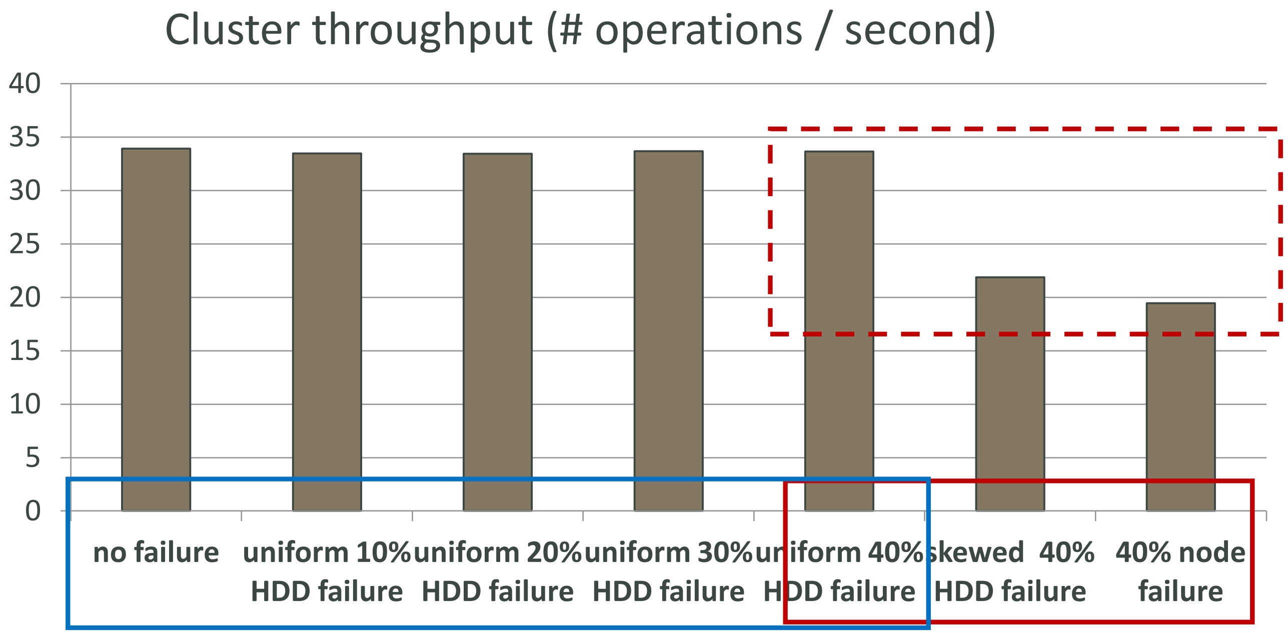


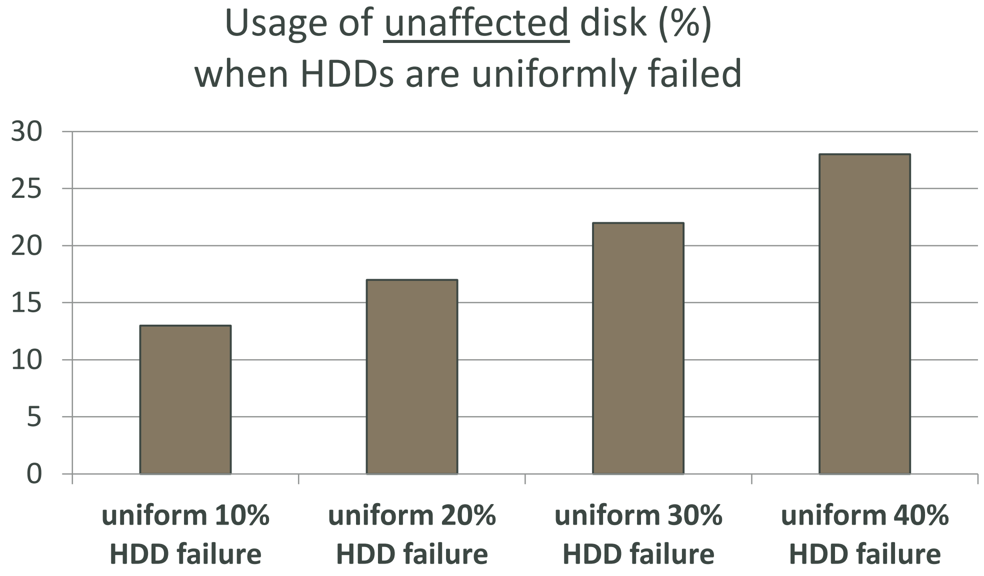
Network bandwidth (MB/s) in unaffected storage node



- **When storage node is based on High-CPU Instance**
  - Over-provisioned resources in unaffected node get more used as # failures increases
  - So, it can keep performance from degrading initially
- **When storage node is based on Large Instance**
  - CPU is almost saturated when no failure happens

Evaluation – HDD Failure





- **When HDD are uniformly failed across all storage nodes**
  - Throughput does not decrease ! Why?
- **When HDDs are uniformly failed**
  - I/O loads are evenly distributed over other unaffected HDDs
- **When some storage nodes have more failed HDDs than others (skewed)**
  - Throughput decreases significantly, still better than entire node failure
  - Extreme case: when all HDDs on a storage node fail, it is almost equal to entire node failure

zmanda
Open Source Backup

Evaluation – Take-away points

- **In order to maintain certain performance in fact of failure**
  - Make sense to “over-provision” the HW resources to some extent
  - When failure happens, the “over-provisioned” resources will reduce the performance degradation
- **Entire storage node failure vs. HDD failure**
  - Entire node failure is the worse than HDD failure.
  - When only HDDs failed, performance degradation depends on:
    - If failed HDDs are uniformly distributed across all storage nodes
      - degradation is smaller, because I/O load can be rebalanced over unaffected HDDs
    - Otherwise (failure distribution is skewed)
      - degradation may be larger
- **What if proxy node failure? proxy and storage nodes fail together?**
  - Reduce the performance, need to figure out in the future

When Failed Nodes Are Fixed

- When all failed (affected) nodes have been fixed and re-join the Swift cluster
  - **(1) How soon the recovery will take on the affected nodes?**
  - (2) What is performance when the recovery is undergoing?

  - We will show empirical results in our blog (http://www.zmanda.com/blogs/)
  - For (1), it depends on:
    - How much data need to be recovered.
    - Networking latency b/w unaffected and affected nodes
    - HW resources (e.g. CPU) in unaffected nodes (lookup which data need to be restored)

zmanda
Open Source Backup

When Failed Nodes Are Fixed

- When all failed nodes have been fixed and re-join the Swift cluster
  - (1) How soon the recovery will take on the affected nodes?
  - **(2) What is performance when the recovery is undergoing?**

  - For (2), it depends on:
    - HW resources in unaffected nodes. The unaffected nodes become more resource-intensive because they still serve requests, also help affected nodes to restore their data

  - Performance will gradually increase as the recovery progress is close to 100%

zmanda
Open Source Backup

Thanks! Questions/Comments?

http://www.zmanda.com/blogs/

swift@zmanda.com

zmanda
Open Source Backup

Back-up Slides

Evaluation – Download small objects

- Top-3 recommended hardware for a small Swift cluster

		HW for proxy node	HW for storage node	Throughput/$
Download Small Objects	**1**	**2 proxy nodes (High-CPU)**	**5 storage nodes (Large)**	737
	2	2 proxy nodes (Cluster)	5 storage nodes (Large)	572
	3	2 proxies nodes (High-CPU)	10 storage nodes (Large)	513

- **Storage node** is all based on **Large**
  - Only one copy of data is retrieved. CPU and disk I/O are not busy
  - Large is sufficient for workload and saves more cost than High-CPU
- **Proxy node**
  - Traffic pattern: high throughput, low network bandwidth (e.g. 2400 op/s -> 117 MB/s)
  - 1 GE from High-CPU is adequate
  - 5 **Large** based storage nodes can almost saturate the 2 **High-CPU** based proxy nodes.

zmanda
Open Source Backup

Evaluation – Download large objects

- Top-3 recommended hardware for a small Swift cluster

		HW for proxy node	HW for storage node	Throughput/$
Download Large Objects	**1**	**2 proxy nodes (Cluster)**	**5 storage nodes (Large)**	16.8
	2	2 proxy nodes (Cluster)	10 storage nodes (Large)	14.5
	3	2 proxy nodes (High-CPU)	5 storage nodes (Large)	12.9

- **Storage node** is all based on **Large**
  - Request rate is low, little load on CPU.
  - Large Instance is sufficient for workload and saves more cost than High-CPU.
- **Proxy node**
  - Traffic pattern: low throughput, high network bandwidth
  - e.g. 70 op/s -> 350 MB/s for incoming and 350 MB/s for outgoing traffics
  - 1 GE from High-CPU is under-provisioned, 10 GE from Cluster is **paid off** for this workload.
  - 5 **Large** based storage nodes can nearly saturate the 2 **Cluster** based proxy nodes.

zmanda
Open Source Backup