BARNS: Backup and Recovery for NoSQL Databases

Atish Kathpal, Priya Sehgal

Advanced Technology Group, NetApp
Why Backup/Restore NoSQL DBs?

Customers are directly ingesting into NoSQL

Security breach are on the rise e.g. ransomware attacks on MongoDB [1] and recent WannaCrypt exploits

“Fat-finger” errors eventually propagate to replicas

Sandbox deployments for test/dev

- Bring up shadow clusters of different cardinality (from production cluster snapshots)

Compliance and regulatory requirements

IDC, 2016 report [2] lists data-protection and retention as one of the top infrastructural requirements for NoSQL

NoSQL Database Classes
From Backup/Restore Service Perspective

- Master-slave
  - Authoritative copy of each partition is contained in the master node that we can backup.
  - Loss of primary node leads to shard/partition-unavailability until new leader is elected.
  - Example: MongoDB, Redis, Oracle NoSQL, MarkLogic

- Master-less
  - Data is scattered across nodes using consistent hashing techniques, no single node has all data for a given partition
  - *Eventual consistency*: Unavailability of a destination node does not lead to write-failure, data is eventually replicated
  - Example: Cassandra, Couchbase

NoSQL DBs Hosted on Shared Storage
High-level, conceptual deployment architecture

Backup: Leverage storage snapshots
Restore: Leverage cloning

Shared Storage Array
(Snapshots, Cloning, Compression, Deduplication, Encryption, Cloud Integrations)
Backup/Restore Challenges

- **Cluster-consistency at scale**
  - Cluster/App quiesce – significantly hampers application performance. Cross node consistency not guaranteed
    - Take crash consistent snapshots
    - Post process crash consistent snapshots (in a sand-box) using NoSQL DB stack to reach an cluster-consistent state

- **Space Efficiency**
  - Replica set data copies do not de-duplicate – small row sizes, scattered across nodes (Cassandra) and unique ids added by storage engines (MongoDB)
  - DB performs compression and encryption
  - Remove replicas logically (application aware backup)

- **Topology Changes**
  - Commodity nodes, at scale of 10-100s of nodes. E.g., Primary node might be unreachable while taking backup in case of MongoDB
  - Storage snapshots do not have context about cluster topology
  - Use cases may require restore to a test/dev cluster of different cardinality
  - Save Cluster topology and storage mapping as part of backup

Existing open source utilities like Mongodump and Cassandra snapshots suffer from above challenges.
NoSQL Data Protection – Challenges

Master-Less Databases

Challenges:

- **Fault tolerance**
  - Backup may capture stale data due to *eventual consistency*
- **Higher restore times**, since Cassandra will perform *repairs* during restore

BARNs Architecture

Addresses challenges of:
1. Taking cluster-consistent backup at scale
2. Taking storage efficient backups (through replica removal)
3. Enables recovery/cloning to different cluster topologies
BARNS Solution: Cassandra

Master-less Distributed Database
Phase 1: Light-weight Backup Phase

Production Cluster

1. Capture token assignment of each node
2. Store mapping of LUNs → Tokens
3. Take snapshots of L1 to L4

Backup Metadata example

```json
{"backup_name": "1488869633.586644", "cluster_name": "barns", "members": [
{"lun": "/vol/cass1/lun_cass1", "snap-name": "17-03-02_01:02:18", "stateStr": "Healthy", "tokens": "<list of tokens> " }]
{"lun": "/vol/cass2/lun_cass2", "snap-name": "", "stateStr": "UnHealthy", "tokens": "<list of tokens> " }]
```
Phase2: Post Process Phase

Part1: Flush Commitlogs

1. Clone the LWB Snapshot LUNs
2. Mount on different PP processes or different nodes
3. Start Cassandra processes P1 to P4

4. Flush CommitLogs

E.g. Data
K1, V1, T1
K2, V2, T1
K1,V11, T2
Phase2: Post Process Phase

Part2: Compaction

1. Using UnionFS mount all snapshot clones as read-only
2. Create a new volume and LUN for full backup
3. Mount it as RW through UnionFS
4. Let PP node have all tokens of prod cluster
5. Start Cassandra
6. Start compaction process
7. Final compacted files will be stored on Fullbackup LUN

Keeping single copy of data in the backup
=> ~66% reduction in backup storage requirements
Cassandra Restore

The post-process step enables cloning to different restore/clone topologies
Evaluation - Cassandra Backup and Restore

Full Backup

- Production cluster
  - 4 nodes
  - 4 iSCSI LUNs
  - Commitlog and SSTables for a node on same LUN
  - Cassandra 4.0

- Post Process Node
  - 2 CPUs
  - 8GB RAM

- YCSB to ingest data

- LWB – <10 secs
- pp-flush - ~40 secs
- pp-compact – time increases by 35-40% → incremental backup
- Restore time – less than ~2 mins (irrespective of cluster size and data set size)
BARNS Solution: MongoDB

Master-Slave Distributed Database

➢ Check the paper or just attend the poster session 😊
Summary

• Tracking replicas and cluster topologies is important for taking backups and performing flexible topology restores

• Existing open-source solutions have several inefficiencies like need for repairs after restore, lack of storage efficiency in backup and poor integrations with shared storage

• Opportunity to provide efficient backup and restore through light-weight snapshots and clones
Thank You.