Percona Backup for MongoDB

Akira Kurogane
Percona
3 - 2 - 1

MongoDB Community Edition
Percona Server for MongoDB
MongoDB Enterprise Edition
Replica Set
Cluster
Percona Backup for MongoDB
Elements of MongoDB Backups
MongoDB oplog

- MongoDB has logical (not physical) replication.
- Visible to db users in "local" db's oplog.rs collection.
- User writes will be transformed to idempotent operations:
  - A write modifying $n$ docs will become $n$ docs in the oplog, each with "_id" value of affected doc.
  - Relative modifications become absolute
    E.g. `{"x": {$inc: 1}} → {"$set": {"x": <newX> }}
  - Nested arrays usually $set$ as whole every modification.
- Transactions pack several ops together for a single apply time.
- Secondaries apply oplog ops with broad-use "applyOps" command.
MongoDB oplog - Extra Use in Backups

A database dump has a phase of copying all collection documents. Let's say this takes $m$ minutes.

- The last dumped doc is as-of time ($T$).
- The first dumped doc is as-of ($T - m$) mins.

Inconsistent! But easy fix to make all docs match time ($T$).

- Get oplog slice for those $m$ mins.
- Replay the (idempotent) oplog on the dump.
Consistency (Replica Set)

All methods below provide consistent snapshots for replica sets:

- Filesystem snapshot method  
  Storage engine's natural consistency
- Stopped secondary  
  Storage engine's natural consistency
- Dump method + oplog slice during copy  
  = reconstructable consistency as-of finish time.

All the DIY scripts or tools use one of the above.

*(But don't forget --oplogFile if using mongodump in own script!)*
Consistency (Cluster)

As for a replica set, but synchronized for all replicaset in cluster:

- Config server replicaset as of $t_x$
- Shard 1 replicaset as of $t_x$
- Shard 2 replicaset as of $t_x$
- ...
Consistency (Cluster)

Concept 'gotcha': Simultaneous-for-everyone consistency impossible.

Network latencies to shards == relativity effect.

2 clients. Far shards with 2ms RTT latency, Near shards with 0.2ms RTT.

- Initiate reads to Far shards at \(-1.5\)ms
- Read happens on Far shards at \(-0.5\)ms
- Initiate writes on Near shards at \(-0.1\)ms
- Writes happen at 0 ms
- Writes confirmed by response +0.1ms
- Reads returned in response at +0.5ms

Both observe the Near write before Far read. Asymmetric.
Consistency (Cluster)

Minimal client latency relativity effect per different point-in-time definitions:

- Same wall-clock time by oplog: Clock skew + RTT.
- Same time according to one client: RTT latency.
- Single client's 'checkpoint' write: Perfect to that client; RTT to others.

All approximately same accuracy, on the scale of milliseconds.

- Very accurate by human response times.
- Crude by storage engine op execution time.
Consistency (Cluster)

Minimal client latency relativity effect by point-in-time definitions:

- Parallel filesystem snapshots: Snapshot op time + RTT.
- Hidden secondary snapshots: Shutdown time + RTT.

"lvcreate -s ..." ~≈ several hundred milliseconds (my experience).

Node shutdown: typically several seconds (my experience).
Point-in-time Restores

Backup snapshot at time $st_1$  \(\implies\)  Restore to any point in time between $st_1$ to $t_x$
Copy of oplog from $\leq st_1$ to $t_x$ \(\implies\)  PITR from $st_{oldest}$ to now.
Daily snaps + 24/7 oplog history

Note:
- Large write churn = too much to stream to backup store. Give up PITR.
- Since v3.6 need to skip some system cache collections: $config.system.sessions$, $config.transactions$, etc.
Transactions - Restore Method

MongoDB 4.0 replica set transactions.
- Appear as one composite oplog doc when the transaction completes. Just replay as soon as encountered when restoring.

MongoDB 4.2 distributed transactions
- In most situations the same as above (w/out 16MB limit).
  Just replay as soon as encountered when restoring.
- Only multi-shard transactions use new oplog format.
- Distributed transaction oplog has separate docs for each op.
- Buffer these and don't replay until "completeTransaction" doc found.
Existing MongoDB Backup Tools
MongoDB Backup Methods (DIY)

mongodump / mongorestore:
Simple ✔️  Shard ✔️  Easy restore ✔️  PITR ✗  S3 store ✗  HW cost $  
or
Simple ✗  Shard ✔️  Easy restore ✗  PITR ✗  S3 store ✗  HW cost $

Filesystem snapshots:
Simple ✗  Shard ✔️  Easy restore ✗  PITR ✗  S3 store ✔️  HW cost $

Hidden secondary:
Simple ✔️  Shard ✔️  Easy restore ✗  PITR ✗  S3 store ✔️  HW cost $
Percona Server for MongoDB has command for hot backup:

```
> use admin
> db.runCommand({createBackup: 1, <local dir or S3 store>})
```

**PSMDB Hot Backup (Non-sharded replica set):**
- Simple ✓
- Sharding ☒
- Easy restore ☒
- PITR ☒
- S3 store ✓
- HW cost $  

**PSMDB Hot Backup (Cluster):**
- Simple ☒
- Sharding ✓
- Easy restore ☒
- PITR ☒
- S3 store ✓
- HW cost $  

(similar to filesystem snapshot, but extra unix admin for LVM etc. avoided)

New in v4.0.12-6
MongoDB Backup Methods (Tools)

MongoDB OpsManager (*Paid license; closed source*)
Simple ☒ Shard ing ☑ Easy restore ☑ PITR ☑ S3 store ☑ HW cost $$

mongodbc onsistent-backup (Percona-Labs repo)
Simple ☑ Shard ing ☑ Easy restore ☑ PITR ☒ S3 store ☑ HW cost $

percona-backup-mongodb v0.5
Simple ☒ Shard ing ☑ Easy restore ☑ PITR ☒ S3 store ☑ HW cost $
MCB; PBM v0.5

mongodb-consistent-backup
- single script
- single-server bottleneck  Not suitable for many-shard clusters

percona-backup-mongodb v0.5
- pbm-agent  1-to-1 to mongod (copy bottleneck gone)
- pbm-coordinator  Coordinator daemon to agents
- pbm CLI

"Simple ☒" because coordinator-to-agents is an extra topology
percona-backup-mongodb v1.0

- pbm-agent  1-to-1 to mongod
- pbm-coordinator  Coordinator daemon to agents
- pbm CLI

Simple ✔️  Sharding ✔️  Easy restore ✔️  PITR ☒  S3 etc. ✔  HW cost $

Now: Manual PITR on restored snapshot is OK  Full Auto PITR is next major feature on dev roadmap
pbm-coordinator (R.I.P.)

percona-backup-mongodb v0.5
- pbm-agent 1-to-1 to mongod
- pbm-coordinator Coordinator daemon to agents
- pbm

Why kill the coordinator ...?
"Let's Have a Coordinator Daemon"

Cluster shard and configsrvr backup oplog slices must reach same time -> Coordination is needed between the agents.

"So let's have a coordinator daemon. We just need:

- One or two more setup steps.
- Extra authentication subsystem for agent <-> coordinators.
- A few more ports open (== firewall reconfig).
- New pbm commands to list/add/remove agents.
- Users must notice coordinator-agent topology first; troubleshooting hard.
"New Idea: Let's Not!"

But how do we coordinate?

REQUIRED: Some sort of distributed server
- Already present on the MongoDB servers.
- Where we can store and update config data.
- Agents can listen for messages as a stream.
- Has an authentication and authorization system.
- Agents can communicate without firewall issues.
- Automatic failover would be a nice-to-have.
- ...
Coordination Channel = MongoDB

**pbm** sends message by updating a pbm command collection.
**pbm-agent**s update their status likewise.

- Already present on the MongoDB servers (duh!)
- Store and update config data in *admin.pbm* collections.
- Agents listen for commands using MongoDB change stream.
- Use the MongoDB authentication and role-based access control.
- Agents connect only to mongod hosts so no firewall reconfig needed.
- Automatic failover provided by MongoDB's replication.
PBM's Collections (as of v1.0)

- `admin database`
  - `pbmCmd` The trigger (and state) of a backup or restore
  - `pbmConfig` Remote store location and access credentials
  - `pbmBackups` Status
  - `pbmOp` Coordination locks
Lose DB cluster, Lose Backup System?

Q. If the cluster (or non-sharded replicaset) is gone, how can the `pbm` command line tool communicate with the agents?

A: It can't.

In the event of a complete loss / rebuild of servers:

- Start a fresh, empty cluster with same RS names.
- Create the `pbm` mongodb user with backup/restore role.
- Re-insert the remote-store config (S3 URL, bucket, etc).
- "`pbm list`" --> backups listed by timestamp.
- Restart the `pbm`-agent processes.
- "`pbm restore <yyyymmdd_hhmmss>`".
Demonstration
Demonstration

pbm --help
pbm [--mongodb-uri ...] set store --config <S3_config.yaml>

pbm-agent --mongodb-uri mongodb://user:pwd@localhost:port/

pbm [--mongodb-uri ...] backup
  (aws s3 ls s3://bucket/...)

pbm [--mongodb-uri ...] list

pbm [--mongodb-uri ...] restore <yyyyymmdd_hhmmss>
Coming Features

- Point-in-time restore.
- `pbm status`, `pbm log`.
- Distributed transaction oplog handling.
Point-in-time Restore

Agents already copy variable length of oplog for cluster snapshots.

"Snapshot" time $= \min(\text{oplog slice finish times})$

- $= 0 \sim \text{few secs after slowest data-copy end time}$

- Agents replay oplog slices only to that snapshot time.
- (Parallel application in each shard and configsvr RS).
Point-in-time Restore

Let's use the same oplog capture and replay functionality. To come as next main feature in PBM:

- Option to add oplog capture 24/7 to enable PITR.
- After restore of backup snapshot at $t_s$ replay oplog from $t_s$ to $t_x$
- (Parallel application in each shard and configsvr RS)
Point-in-time Restore

**Manual PITR** is already possible on top of a PMB v1.0-restored backup if
- The cluster isn't already erased, and;
- The oplog(s) start before that backup's time.

Method:
1. Dump the oplog(s) elsewhere **before** doing "pbm restore"
2. Use mongorestore --oplogReplay --oplogFile ....

User Interface

`pbm status`
Show the progress of running backups

`pbm log`
Centralized agent log display
Transaction Consistency Now

Transactions consistency supported by PBM so far (v0.5, v1.0)
- 4.0 Replica set transactions.
- 4.2 Single shard-affecting transactions.

Mechanism for these transactions:
- MongoDB creates single oplog doc at commit time.
- Transaction's write ops wrapped in a nested "applyOps" array.
- Just apply as the next op, like classic oplog mechanism.

Not unique to PBM. `mongorestore` can restore these too.
{  "ts" : Timestamp(1567058020, 1),  ...
  "op" : "c",
  "ns" : "admin.$cmd",
  ...
  "txnNumber" : NumberLong(2),
  ...
  "o" : {
    "applyOps" : [
      {
        "op" : "i",
        "ns" : "test.baz",
        "ui" : UUID("54b05710-ee45-4cca-9bd1-63b749ed6557"),
        "o" : { "_id" : ObjectId("5d676859138f17a8d8a27bb8") } },
      {
        "op" : "i",
        "ns" : "test.bar",
        "ui" : UUID("5c65df08-da5e-4ef8-8bb0-27bfa3b50c80"),
        "o" : { "_id" : ObjectId("5d67685f138f17a8d8a27bb9") } }
    ]
  }
}
4.2 Distributed Transactions

Transactions not supported so far (<= v1.0)
- 4.2 Multiple shard-affecting transactions.

Mechanism:
- Transaction ops written separately ({{.., "txnNumber": ..., {{.., "prepare": true}}}).
- Don't apply immediately. Buffer in chain for that txn.
- Apply all when 'completeTransaction' reached.
- Discard buffered ops if 'abortTransaction', or if replay simply finishes.
{ "ts" : Timestamp(1567134752, 6),
  ...
  "op" : "d",
  "ns" : "config.transaction_coordinators",
  ...
  "o" : {
    "_id" : {
      "lsid" : {
        "id" : UUID("995ad9a8-9d95-43c5-acbe-1a987df4fc95"),
        "uid" : BinData(0,"kanlvzjTP1bYGUTMfQK71txdM8LpbSXTMtQ+b8M4WTA=")
      },
      "txnNumber" : NumberLong(0)
    }
  }
}
4.2 Distributed Transactions

Backup tools supporting 4.2 Distributed Transactions as of now.

Needed only if your backup snapshot time bisects multi-shard transactions.

- MongoDB Ops Manager v4.2
- mongodump + mongorestore
- Filesystem snapshot method
- Percona Backup for MongoDB v1.0

Roadmap: Percona Backup for MongoDB to be PITR in v1.2.