Data Model – Configuration Data

# Introduction

This document describes how ExaVenger manager manages the configuration and status of a ExaVenger cluster. To achieve high reliability, we use multiple manager nodes backed up by a Zookeeper ensemble.

Per our Partition Map Design, the partition map is cached in the client broker as a whole. To save memory, the partition map **part\_map** is an array of 1 byte sequence numbers. The server map **ser\_map** is an array of server addresses (4 byte IP and 2 byte port). To locate the server for a partition **X**, we first get the sequence number **S = part\_map[X]**. We use the partition ID and the sequence to obtain an index to the server map: **I = hash(S, X) % Num\_servers**. Then we can obtain server address: **ser\_map[I]**.

The idea is to give each partition 256 servers to choose hosts from. We need to choose a hash function so that host sets of different partitions only partially overlap. Suppose each host can support 300 partition at a time, during fail-over, we want to have at least 600 machines to distribute these partition to. A tentative choice of the hash function is **CRC32(S \* Num\_partitions + X)**.

One tricky point is that the manager usually needs to modify multiple zookeeper nodes in reaction to a single event (such as a machine dies). In the case of the manager crashes in the middle of the event processing, how can we make sure the new manager can always retry what’s left unfinished.

We are evaluating two ideas. One is using the constant stream of error report from the client broker to resume the actions left unfinished by a crashed manager. Another is to write a transaction log first, perform the action and then close the log entry.

# Configuration data definition for Zookeeper

## Structure

Here we discuss the structure of znodes and the data associated with them

\ExaConfig

\PartitionMap

\<partition id> array of 3 sequenceNums which will map to index in server map

…

\ServerInfo

\<server address> Server status, indexes of server map

…

\DefectiveServers

\<server address> complains from client broker

…

## PartitionMap

This is different from the in-memory partition map that we maintain in ExaBroker.

|  |  |
| --- | --- |
| Parent Node | /partitionMap |
| Children Node | /<patitionId> |
| Node Data Definition | std::array of 3 sequenceNums which will map to index in server map (1 seqNum for one partition) |

## ServerInfo

Parent Node definition:

|  |  |
| --- | --- |
| ParentNode | /serverInfo |
| Children Node | /<serverId> |
| Node Data Definition | [ServerBasicInfo](#_MachineServer_Info) |

## Defective Server List

Parent Node definition:

|  |  |
| --- | --- |
| ParentNode | /defectiveServers |
| Children Node | /<serverId> |
| Node Data Defnition | Number of complaints  Timestamp of the last registered complaint |

# In-memory Objects

This section describes the in-memory representation of the configuration data that we maintain for ExaVenger Store

## Server Address Object



|  |  |  |
| --- | --- | --- |
| **Class** | **Member** | **Notes** |
| ServerAddress |  | Structure to store server address info |
|  | ServiceIPv4 | The IPv4 address of the server.  We are designed exclusively to run within the DC, where IPv4 will persist. |
|  | ServicePort | The Port on the server where the EBServer is listening on. |

## Server Capacity Object



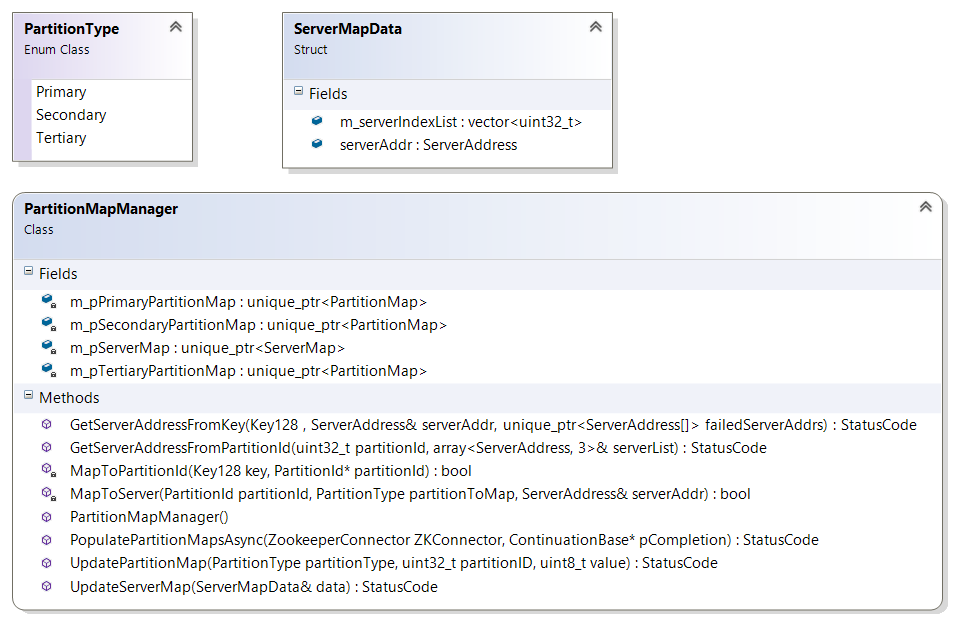
|  |  |  |
| --- | --- | --- |
| **Class** | **Member** | **Notes** |
| ServerCapacity |  | Structure to store server capacity info. When a server becomes alive and reports to the manager, it sends its capacity information to the manager |
|  | MaxPossibleHDDPartitions | Maximum number of Hard Disk Partitions this server can host. |
|  | MaxPossibleSSDPartitions | Maximum number of SSD Partitions this server can host. |

## Partition Map Manager

This is an in-memory logical representation of partition map. PartitionMapManager stores all the 3 partition maps. This is cached on ExaBrokers to direct the requested key to the correct server

which is hosting the key's partition

**Partition Map** is defined as an array of 1 byte integers which carry a sequence number between 0 to 255. We use this value to get the server index in the ServerMap. The length of the partition map equals the number of partitions. Note: Number of partitions are predetermined.

**ServerMap** is defined as array of server addresses. Each live server appears at least once in this map  
  


|  |  |
| --- | --- |
| **Member** | **Notes** |
|  |  |
| m\_pPrimaryPartitionMap | This is unique pointer to an array of length == numOfPartitions which stores a sequence number which is used to get the index of server in the server map |
| m\_pSecondaryPartitionMap | Same as above – holds the sequence number for secondary partition |
| m\_pTertiaryPartitionMap | Same as above – holds the sequence number for tertiary partition |

|  |  |
| --- | --- |
| **Operations** | **Notes** |
| GetServerAddressFromKey | Returns the address of the server holding the primary partition for the requested key if the failedServerAddrs array contains a match for this partition then it finds the address for the secondary server and if this match is also present then this method will return the tertiary partition holder server address. Key to partition id mapping is done using a stab;e hash function. |
| GetServerAddressFromPartitionId | Returns an array of 3 servers which hosts the requested partition id |
| UpdatePartitionMap | This will be called by WatchContinuation when the zookeeper watch event is triggered.  partitionType - The partition table to update  partitionID - The partition ID to update  value - actually data to be updated |
| UpdateServerMap | This will be called by the WatchContinuation when the zookeeper watch event is triggered |
| PopulatePartitionMapsAsync | On startup broker will populate its partition map from zookeeper. This is a startup cost. Once the map is populated future updates will happen through zookeeper watch event. After all the maps are populated the pCompletion continuation will be triggered |
| MapToPartitionId | Hashes the key to get the partition id to which this key belongs to |
| MapToServer | Finds the server address which holds the partition ID for the requested partition type |

## Machine\Server Info



|  |  |  |
| --- | --- | --- |
| **Class** | **Member** | **Notes** |
| ServerBasicInfo |  | Holds the basic info for the server |
|  | m\_address | ServerAddress see section 2.1 |
|  | m\_capacity | ServerCapacity explained in section 2.2 |
|  | m\_chainID | Chain to which this machine belongs. |
|  | m\_status | Machine Status |
|  | m\_complaints | Number of complaints received for this server. Initially this is 0 |
| ServerInfo |  | Maintains Complete server information along with the list of partition the server hosts |
|  | m\_primaryPartitionList | List of primary partitions hosted by this server |
|  | m\_secondaryPartitionList | Same as above for secondary list |
|  | m\_tertiaryPartitionList | Same as above or tertiary list |

## Server Info Manager (Not edited yet)

This class defines operations for managing all the info related to the servers (active + reserved) in the current operation pool.



# Data Flow between Manager, Broker and Zookeeper

## EBServer Node Startup

* On Startup EBServer node will report to EBManager with its basic info
  + Address
  + Capacity
* The *ServerInfoManager* in EBManager will add this server to the in-memory server info map and to the ServerMap.
* It then creates a zookeeper server info node or update the existing node with status = reserved (/serverInfo/<serverId>) and then send an ACK to the requesting EBServer
* On adding this data to ZK either of the two watches NewNodeCreated\NodeDataChanged is triggered for /serverInfo/<serverId> node, backup EBManagers pick up this change and update their local copies
* Brokers are also triggered. Brokers update their local copy of server map based on this info
* The manager will send a reply after only after it finishes processing. The EBServer will keep retrying until receiving a reply. In the case where manager crashes, the new manager will receive EBServer’s retry and redo the whole thing.

## Fail-over

ExaBroker sends complain to EBManager if a EBServer timeout,

EBManager decides whether to mark the server as dead based on the complains, and removes the server from server map, adds all partitions served by this server to other servers (other servers are determined by the CRC sequence number).

If the manager crashes while doing this. Since the complaints from brokers keep coming, the new manager can retry.

Another case is that the manager receives a “new start” message from a server that is currently serving partitions. In the case the server must have restarted, and we need to assign its partitions to other machines. Since the server will retry this “new start” message, we are safe from manager crashes.

### Marking a Server as Dead

We need to keep track of complains to one server in Zookeeper, for manager fail-over. One difficulty here is that the complaints might be coming at a fast rate that we need to somehow group the process of these complains.

Two criteria must be met for a server to be marked as dead: complains coming from multiple (> 5) sources, and they must last for more than some time. We need an algorithm for this.

### EBServer Remove from ServerMap

When a server is marked as Dead the PartitionMapManager replaces the offset which holds this server address in ServerMap with an "Invalid" entry.

### Assign Partitions to Servers

* As EBManager encounters a server machine failure (receive new start or dying msg from server, non-responsive for more than 5 min), it needs to assign partitions served by the dead machine to others.
  + EBmanager maps partition to a server ServerId = (Sequence num \* #partitions + PartitionID) % Length of Server Map
  + Updates the local copy of Server to partition map (This is a list of all partitions hosted by this server)
  + Updates the local copy of the Partition Map
  + Updates the ZKNode for partition map .i.e /PartitionMap/<partitionId>
* Once the ZKNode for PartitionMap is updated the Brokers get this watch event
  + - PartitionMapManager issues a getData call
    - Foreach server from the 3 it retrieves the sequence number
    - And updates its partition tables accordingly

## Populating PartitionMap and ServerMap on ExaBroker startup

* On startup PartitionMapManger issues getChildren call to ZK node /partionMap
* Foreach child present in the list it issues getData call. The data received contains an array of 3 sequence numbers [0 – primary partition sequence Num, 1 – secondary partition sequence number, 2 – tertiary partition sequence number]. It updates all the three tables based on this.
* It also issues getChildren call to ZK node /serverinfo
* Foreach child present in the list it issues getData call. The data received contains an array of index where this server belongs. It updates the ServerMap based on this info

## Populating ServerInfo on EBManager Startup

### During system startup

* On system start up this map is empty
* ~~EBManager (leader) creates the default parent nodes~~
* EBManager populates its in-memory structures from the ZK node. Initial structures in ZK node will be ready before the EBManager starts up
* Note: Server Map is not maintained in Zookeeper. EBManager creates an in-memory server map based on the index list present in the ServerInfo ZKNode
* Other EBManagers (non-leaders) set watches on ZK nodes and pick up changes as watches are triggered
* Rest of the servers are added as they start reporting live status

### System is up and running a new manager joins and is starting up

* ServerInfoManager issues a getChildren call to ZKNode /serverInfoMap
* For each server in the list it issues a getData call and updates its server map
* Each server in the list with complaint count > 0 is also added to the defective server list and the partition data associated with this server is pulled from ZK node

# TODO:

1. Define
   1. Zone Id
   2. Server Id – We can use ServerAddress as the key\Id
   3. Max number of chains (should we have a constant?)
   4. Number of partitions (how to decide this – will this be statically known at runtime through some configuration file?)
2. Registering complaints for a server – we need to add timestamp too along with number of complaints as
3. Migrate partition, moving from one machine to another.
   1. This may be necessary for load balancing, but it can wait
   2. Need a data copy protocol
   3. Need to update two server info node, and two partition map nodes.
4. During fail-over a partition say "x" is mapped to a new server say from server 2 it is now mapped to server 6
   1. Once the partition node on ZK is updated a watch is triggered which EBBrokers pick up and update their partition maps locally
   2. Now it so happens that EBBroker 1 was restarting or having some issue because of which it did not process the trigger and now it holds the stale copy
   3. Now if this broker receives read\write requests which are for partition x then it will keep complaining to the Manager that server 2 is not responding
   4. On manager it already knows that 2 is dead but how does it convey the new server which hosts this partition to the complaining broker?
   5. We need to take such cases in consideration