Name:Prayag Chandran Nirmala

SBU ID: 110104135

Design Document: Performance Testing of Apache Zookeeper

INTRODUCTION

My project is to measure performance of Apache ZooKeeper for various application scenarios. Following are the scenarios considered[1]:

1. A distributed file system simulation
2. A messaging system simulation
3. A distributed exclusive locking system simulation

My plan is to implement all these simulations as separate classes and have a driver configure and run tests appropriately.

ZOOKEEPER - AN OVERVIEW[3][5]

Apache Zookeeper offers distributed consensus as a service in a cluster of machines. Any distributed application can use the service by interacting with Zookeeper via a relatively simple API. Now, applications can rely on Zookeeper's implementation of the consensus protocol rather than re-implementing it.

ZooKeeper implements ZAB consensus protocol. Performance is crucial for Apache Zookeeper since any client service which uses Zookeeper is highly dependent on it. For example, Apache Kafka uses Zookeeper to keep track of brokers, consumers, load rebalancing, keeping track of consumed offsets for each partition etc. A performance problem in Zookeeper can severely affect the throughput of Kafka.

ZOOKEEPER - PROGRAMMING INTERFACE[3][4]

Exposes a file system like API to the programmer. A 'ZNode' is roughly equivalent to a file. You can perform operations like create, write, delete etc. on a ZNode. You can also set watches on ZNodes. Watches are a way of listening for changes on a ZNode. ZooKeeper calls a specified callback function when a watched event happens on the watched ZNode. ZooKeeper also has an option to create 'Ephemeral ZNodes'. Ephemeral ZNodes get deleted when the session which created them expires. Another option for a ZNode is 'Sequence

ZNode'. Sequence ZNode option let the programmer create unique names for ZNodes sequentially. This feature is very helpful in realizing many ZooKeeper use cases.

WORKLOAD RECIPES[1][2]

This section provides a brief description of various recipes used to create the required workloads.

FILE SYSTEM RECIPE:

This is a straightforward use of various ZooKeeper API features. The metrics measured are as follows:

- Create throughput: Used the create API call to create ZNodes and measure performance.
- Read throughput: Used the API to read the created ZNodes' data
- Write throughput: Used the API to write data to created ZNodes
- Delete throughput: Used the API to delete the created ZNodes

DISTRIBUTED QUEUE RECIPE:

My implementation has multiple Consumers and Producers. The skeleton of the recipe is as follows:

Producer:

- Create ZNodes in a sequence, with the message as data
- If a ZNode is already present, try next message sequence number

Consumer:

- Read all messages according to the message sequence number
- Set watch on future messages to get notified

DISTRIBUTED LOCK RECIPE:

Following is a quick overview of the recipe for creating a distributed lock:

- All clients create 'ephemeral + sequence' nodes under the root node
- All clients do a getchildren on the root node
- The client with the lowest sequence number holds the lock
- All the other clients set watch on its previous sequence number node Clients get woken up when the previous client releases the lock
- A client releases the lock by deleting its Znode

DESIGN OVERVIEW[3][4]

The driver class takes the hostname and the port number of the ZooKeeper instance, and the number of threads to execute. For each simulation multiple clients are created using threads,

and all execute in parallel. Each thread reports its statistics to the corresponding Runner class. The Runner class aggregates the statistics and prints them out.

Core Classes:[3][4]

- Testing: Driver class for all tests. Runs all tests deterministically, one after the other.
- LockClient: LockClient class to simulate distributed locking.
- LockRunner: Runner class for distributed locking simulation.
- QRunner: Runner class for the distributed queue simulation.
- SyncReadWrite: ReadWrite test class. All tests are tightly coupled as one test.
- SyncReadWriteRunner: The Runner class for distributed file system simulation tests.
- ZkQConsumer: Distributed Queue consumer. Meant to be run in a thread.
- ZkQProducer: Distributed Queue producer. Meant to be run in a thread.

Core Interfaces:

- QConsumer: Every consumer must implement this.
- QProducer: Every producer must implement this.
- RunnableTest: Interface for a Runnable Test. Every test must implement this to be run by the driver.

DETAILED DESIGN

Detailed documentation generated using JavaDoc tool is attached with the submission. Please see doc/index.html under the project source which is attached with the submission.

CURRENT STATUS

Multithreaded implementations of distributed file system simulation and distributed queue simulation are finished. I am currently working on distributed lock simulation. I also plan to change the driver program a bit in the end, and make it more configurable by the user.

REFERENCES:

1. ZooKeeper Recipes: https://zookeeper.apache.org/doc/trunk/recipes.html, Apache Software Foundation, 10/08/2014
2. Source files for ZooKeeper Recipes: https://github.com/apache/zookeeper/tree/trunk/src/recipes, Apache Software Foundation, 07/07/2015
3. ZooKeeper Programmer's Guide: https://zookeeper.apache.org/doc/trunk/zookeeperProgrammers.html, Apache Software Foundation, 10/08/2014

4. ZooKeeper API Docs: https://zookeeper.apache.org/doc/r3.4.5/api/org/apache/zookeeper/ZooKeeper.html, Apache Software Foundation, 11/05/2012
5. ZooKeeper Site: http://zookeeper.apache.org/, Apache Software Foundation