# Handling Mutual Exclusion in a Distributed Application through Zookeeper

Lipika Bose Goel
IMS Engineering College ,
Ghaziabad, India
Lipika.bose@gmail.com

Rana Majumdar
Amity School Of Engineering & Technology
Amity University, Noida, India
rmajumdar@amity.edu

***Abstract*—Zookeeper a powerful, feasible approach to build distributed applications implementing open APIs that enables developers to apply their own powerful co-ordination primitives. The aim of this study is twofold i) To study the anatomy and life cycle of zookeeper and make use of as a role of high-performance coordination service for distributed applications ii) a case study was presented about Zookeeper implementations in the payment process where synchronization and coordination is not meet due to dual server implementation, where orders are placed through an estore application. Finally the potentiality of Zookeeper replications is considered to address reliability and performance issues.**

***Keyword—Zookeeper; znode; mutual exclusion; distributed application.***

## I. INTRODUCTION

A distributed system consists of multiple computers that communicate through a computer network and interact with each other to achieve a common goal. Zookeeper is an open-source server which enables highly reliable distributed applications. It facilitates wait-free co-ordination in highly distributed internet scale systems where mutual exclusion is the prime concern. It is a way to implement mutual exclusion to a shared system. It allows a distributed application to coordinate with each other through a shared hierarchal namespace which is very similar to the standard file system. The name space consists of znodes similar to files and directories. Each node of zookeeper is allowed to have data associated with it along with its children.

### *A. Zookeeper supports the following API:*

- Create : creates a node at a location in the tree
- Delete : deletes a node
- Exists: checks if a node exists at a location
- GetData : reads the data from a node
- SetData : writes data to a node
- Get Children : retrieves a list of children of a node
- Sync : waits for data to be propagated

### *B. Anatomy of Zookeeper:*

Zookeeper involves the interplay of 3 fundamental entities as it offers wait-free co-ordination service for distributed applications:

- Clients are the machines which use Zookeeper service.
- Server denotes the service which uses Zookeeper service.
- Znode is the memory data node in namespace hierarchy.

### C. *Life Cycle of Zookeeper*:

- Initiation: It creates a batch of the pending jobs. Then takes one job at a time and checks its status, In case the status of the task is not complete, it creates a parent node and child node for the pending tasks.
- Execution: Creation of the child node implies the task no. pertaining to it undergoes processing. Post creation of the child node, Zookeeper applies a lock to it until the task gets processed completely.
- Conclusion: Once task is processed completely, child node pertaining to the completed task is deleted .The creation of multiple nodes and locking ensures wait-free processing of individual tasks in a mutually exclusive way.

Zookeeper derives its robustness from a suite of reliable distributed system techniques and protocols and runs on a cluster of machines to provide seamless service.

978-1-4673-6911-4/15/$31.00©2015 IEEE

## II. METHODOLOGY

*Problem Conceptualization :*

### *A. State of affairs*

Customers can purchase products from an estore site developed by the software developing company. On the purchase of products the settlement of the payment process was handled on a different node called settlement node. The application and the settlement node were running on two different servers on production for the purpose of load distribution. Data synchronization/ streaming were maintained between these two servers. In the settlement node the schedulers picked orders from Database which were pending for settlement, meaning that the orders which were not processed for payment. The schedulers then create a batch of unprocessed orders and process them. After processing of the orders its status changes to CLOSED in the database. The next time when schedulers runs it excludes those orders which are in CLOSED state.

### *B. Problem sphere*

As it was stated in the previous section that the presence of settlement node on both the servers, forcing data streaming and schedulers to run on both of them , causes a major issue. The schedulers running on different servers in some cases picked up the same order for settlement thereby the customer was billed twice. Below is the pictorial representation of the above explained problem sphere:

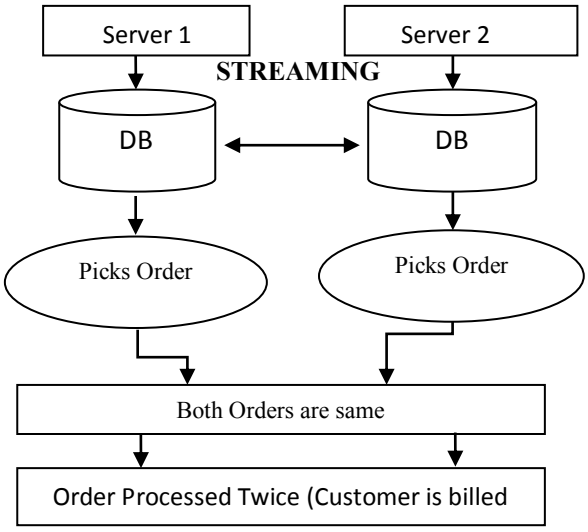


Fig 1. Problem sphere

*1) Processs Adopted:* A Zookeeper environment prevents such instances by ensuring synchronization and application of distributed locks to ensure mutual exclusion to an order under processing.

During the settlement for pending orders, scheduler checks database for the orders pending for settlement. It creates a batch of pending orders.

It takes one order at a time, checks the status of the order (whether it is CLOSED or not) if the status is not closed: Zookeeper creates a parent node for the first time under which all other child nodes will be created. It treats each order as a child node. After creation of the parent node it checks that whether the child node for that order already exists or not. If the node already exists then it restricts to create any further node for that order else it creates a new child node under the parent node. On creation of the child node that specific order number undergoes further processing and settlement (payment related). On settlement of the order the status of the order is updated to 'CLOSED'. The Zookeeper deletes the created child node for that order.

The above mentioned steps assist in mutual exclusion of each order thereby preventing the customer from being billed twice. For instance settlement job is executing in two different Servers. There might be a scenario when both the schedulers running on different servers picks up the same order pending for settlement for further processing. This will cause the customer to be billed twice. Zookeeper server will prevent this by its feature of mutual exclusion. Once an order is picked up for settlement, the Zookeeper will create a node for it and will provide a lock to it until it gets processed. Once processed it will release the lock and the node will be deleted .If in case the same order is picked up by some other settlement job running on different Server then it will not be able to create a node for it and further process it, as the node already exists in Zookeeper for that order.

Below is the pictorial representation and flowchart for the above explained scenario:

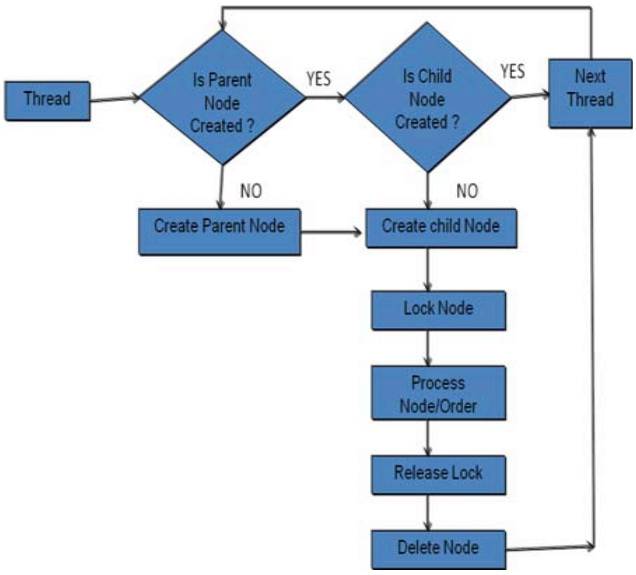


Fig 2. Zookeeper Process.

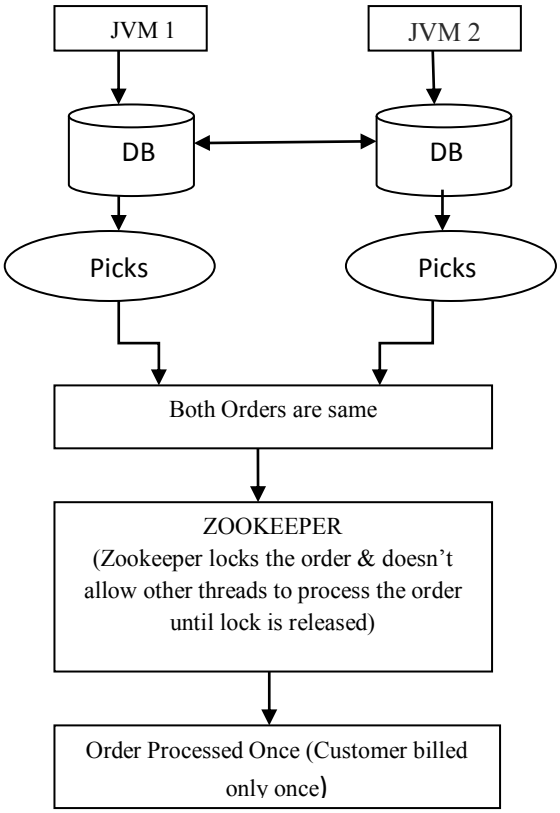


Fig 3. Implementation of Zookeeper

## III. CODING & IMPLEMENTATION

### *A. Steps Followed:*

- Stable version of Zookeeper can be download from the following link: http://zookeeper.apache.org/releases.html
- Install the server at our Application.
- Make changes to zoo.cfg file in conf folder Change dataDir to a location at webServer where snapshots will be created.
- Change java_home entry in 'zkServer.sh' Add export JAVA_HOME=<Insert app's JAVA_HOME> at top of file.
- Go to /bin & run './zkServer.sh start'
- To check server status run './zkServer.sh status'
- Add zookeeper maven plug-in to your Application.
- Paste zookeeper jar to the folder where all the libraries are present.

### *B. Pseudo Code of Zookeeper Implementation in described problem sphere :*

1. Created batch of pending orders.
2. Takes one order at a time checked the status of the Order.
3. We checked if childNode for that order already exists (Boolean isChildCreated=createZookeeperNodeForOrderId (orderUid)).
4. In createZookeeperNodeForOrderId Implementation: First create a new Zookeeper object,then create child node as follows:

String childNode = PARENT_NODE + "/"+orderUid;
Boolean isChildExists = Boolean.FALSE;

Then we check if parent Node exists or not by calling exists() API of Zookeeper:
if (null == zooKeeper.exists(PARENT_NODE, false)) then we create a parent node else parent node already exists.

Next we check if child node exists or not by calling exists() API of Zookeeper:
if (null == zooKeeper.exists(childNode, false))

then we create a child node by calling create () API of Zookeeper:

String childPath = zooKeeper.create();
Set isChildExists = Boolean.TRUE;
else
Set isChildExists = Boolean.FALSE;
return isChildExists;
5. We check if(isChildCreated) then:
Implementation for processing of pending order.
6. Finally we delete the child Node created once the Status of the Order is changed to CLOSED:
deleteChildNode(orderUid);
7. In deleteChildNode Implementation:
We check if (zooKeeper != null)
String childNode = PARENT_NODE +"/"+orderUid;
Check if childNode exists
if (null != zooKeeper.exists(childNode, false))
Call the delete API of Zookeeper:
zooKeeper.delete (childNode, -1)

## IV. RESULT

By implementation of Zookeeper on our Application the major problem of customer being billed twice was resolved. The pending orders were processed only once.

Once a thread picks up an order for settlement, the zookeeper created a node called znode for that task and assigned a lock to it and then processing of the order was done. Once the order was processed the znode was deleted and the state of the order was changed in the database. Even if the same order was picked by some other thread it did not entered into the processing section as the znode for that task already exists.

Mutual exclusion was implemented in the distributed application in a way. Since the order was processed only once customer was billed only once. The process adopted gave a solution to our problem sphere.

## V. CONCLUSION

A Zookeeper environment ensures synchronization and application of distributed locks to guarantee mutual exclusion to the processes. It is an excellent co-ordination service. The creation of the znode, assignation, releasing of the lock resembles semaphore which helps in many Mutual Exclusion problem. The critical section can be accessed by only one process at a time. Running Zookeeper in standalone mode is convenient for evaluation, some development, and testing. Based on our problem statement we created single Zookeeper server although replication of the same process can create quorums. In future replication of the Zookeeper servers will be considered to address reliability and performance issues. Even on the failure of one will not interrupt the process as it will take over by other. Here the perception is to use Zookeeper as “fast-fail”, meaning if a process exists abnormally during it operational phase then it will automatically be restarted and will rejoin the quorum for providing uninterrupted services..

## REFERENCES

[1] Hadoop: http://zookeeper.apache.org

[2] Ranganath Atreya “A Quorum-Based Group Mutual Exclusion Algorithm for a Distributed System with Dynamic Group Set”, Parallel and Distributed Systems, IEEE,18 , Issue: 10, Oct. 2007.pp 1345 – 1360.

[3] T. Chandra, R. Griesemer, and J. Redstone. Paxos made live: An engineering perspective. In Proceedings of the 26th annual ACM symposium on Principles of distributed computing (PODC), Aug.2007.

[4] M. Aguilera, A. Merchant, M. Shah, A. Veitch, and C. Karamanolis. Sinfonia: A new paradigm for building scalable distributed systems. In SOSP ’07: Proceedings of the 21st ACM symposium on Operating systems principles, New York, NY, 2007.

[5] Yoshida, E. ”A learning system for the problem of mutual exclusion in multithreaded programming”. Advanced Learning Technologies, 2004. Proceedings. IEEE International Conference, 30 Aug.-1 Sept. 2004,pp 2 - 6

[6] A. Hastings. Distributed lock management in a transaction processing environment. In Proceedings of IEEE 9th Symposium on Reliable Distributed Systems, Oct. 1990.