SquirrelDB Project Proposal

California Polytechnic State University
Nick Alereza, Stephen Calabrese, Nick Clarke,
Ethan Frame, Grant Frame, Luke Plewa

Abstract

The purpose of a database is to store large amounts information for retrieval and usage in the future. Most database implementations store data on disk. Our database implementation will use an alternative approach to storing data. Data will be stored in main memory. This move will improve the overall speed of the database, at the sacrifice of overall storage space. To add more storage we will be implementing a distributed database; utilizing a collection of nodes to store the data and thus increase the overall capacity. One of the nodes will be considered the master node and all the others slave nodes. Each slave node will only be able to communicate with the master, creating a hub and spoke scheme. In order to remove the bottleneck of message passing, the master node will store as much metadata about the data in the database it can. Another salient feature in our architecture is the lookup table contained in the master node. This will effectively be the (key, (node, memLoc), (node, memLoc), dataLen) tuple. For each (key, value) the master knows which node holds the data, the memory location of that data on that node, and the length of the data. Such design decisions have been made to ensure the database is as quick as possible, sacrificing consistency and overall memory size in the process. Mechanisms have also been implemented to ensure continued functionality in case the master node crashes.

1 Project Topic Desciption

SquirrelDB is a key-value database in memory using a slave/master multi-node distribution written in C. It will use a query API consisting of four functions: get, put, update and delete. It will also utilize the disk only periodically as backup. The nodes will be divided such that there is one master node, which does computations, stores the index table for the database and tracks the amount of free space for the slave nodes, and slave nodes connected to it which actually store the data. The communication between nodes will utilize the Open MPI library. When storing data, SquirrelDB will store every value in two nodes to ensure data availability in case a node becomes unusable for whatever reason.

2 Project Architecture Diagram

Last page of proposal.

3 Component Overview

Listed below are descriptions of the major components of the database broken up by which node in the system the component lives.

3.1 Master Node

The master node will be responsible for several components of the database such as, the transaction manager, the crash recovery system, and the logging system.

Transaction Manager

When new data is to be stored the master node will consult its table of available memory per node and it will choose two slaves to store the data. It will attempt to store the data on those two nodes. If it fails to write to both then it will, assuming it wrote to the other, remove that data from the other and return an error. Upon a successful insert of the data, some meta-data will be stored in the master node so that retrieval of such data is a quick as possible. The key for the data will be fed to a hashing function which will produce an value corresponding to an index where the data should be stored in the master's meta-data list. Thus, the meta-data can be quickly found upon future transactions.

When data is to be deleted the transaction manager will try and delete the data from the two nodes that are storing it. If it fails to remove the data from one or both of the nodes, the data will be restored to the node that deleted it and then an error will be returned.

When data is to be retrieved from the database the transaction manager will consult a list of which node has the data corresponding to which key. The transaction managers job will be to ask the two nodes which have the data for the data. If it doesn't get a response then an error will be returned.

When data is to be updated the transaction manager first tries to get the data from the two nodes that are storing it. Then it will delete the old copy of the data. Finally it will try and put the new data in. If at any point a failure occurs

the transaction manager will restore the state of the database to where it was before the update transaction occurred and return an error.

Crash Recovery System

Every t minutes both the master node and each slave node will write its data to disk, if a change has been made in the last t minutes. If a crash occurs, then the crash recovery system will carry out all the transactions that have been logged since the last time everything was written to disk. Also the master node will flush it's meta-data tables to each slave node to be written to disk. In the event that the master node crashes, one of the remaining slave nodes will be promoted to master. It will then load the meta-data passed by the previous master; trying to resolve conflicts in the meta-data, such as what to do with the data that was previously stored by itself.

Logging System

Write ahead logging will be used by the database to make sure that the current state is the correct one. Logging will be done only by the master node and written to every t minutes, if something was logged in the past t minutes. The log will also be written to all slave nodes, so that they have it in case the master crashes.

3.2 Slave Nodes

The slave nodes will respond to the request that the master sends. The slave nodes hold all the data in the system. The slaves will only know to send data to the master when a request comes in. They do not know anything else about the state of the system. They will periodically back up their current state to the disk. If there is ever a discrepancy between two slaves data the master will send a request to update the slave it believes is out of sync. The data on the slave nodes will be broken up into blocks where the first four bytes are a timestamp for the data that follows in that block. If a piece of data spans multiple blocks then the time field will be left empty, and only the last block with data will get a timestamp. This timestamp will then be returned to the master node when it queries the slave for data, so that the most up to date data is returned to the user.

4 Query Functionality

For this project we will be implementing a query API consisting of four functions: get(key), put(key, value), update(key, value), and delete(key).

4.1 get(key)

Using the key passed in the master will look up if it is stored in the database. If so then the most recent version of the data will be returned to the user.

4.2 put(key, value)

The master will first check and see if the value is smaller in size than the max size a value can have (TBD), and return an error if it is too large. Then it will check to see if the key is already in the database, if so then an error will be returned. If not then the master will check which two slave nodes have the most free space. If there are two nodes which can hold the value it will be stored in the database. If no room is left then an error will be returned.

4.3 update(key, value)

The master will check to see if key is actually stored in database. If not, an error is returned. If yes, the master will delete the old entry and attempt to put the new entry in. If it fails then the old value will be restored and an error will be sent back.

4.4 delete(key)

The master will check to see if key is in the database. If no, an error will be returned. If yes, then all data associated with the key will be deleted.

5 Implementation Notes

5.1 Language of Implementation

We will be using the C programming language. All teammates are comfortable with it and it allows us the most direct control over the system.

5.2 Choice of Distribution Model

SquirrelDB uses a master/slave node setup and will utilize as many slave nodes as possible.

The master node is responsible for the transaction manager, crash recovery system, and the logging system. The master node holds 2 index tables. The first table maps keys to slave nodes and memory locations (start and length) so that data can be easily found based on the key, with as few messages passed as possible. The second table holds a list of available memory space for each node.

The slave nodes are lightweight. They only hold data. Periodically they will back up their data to local disk.

5.3 Tools

SquirrelDB uses the Open MPI library. This library allows usage of message passing functions. According to the Open MPI website, ”MPI is a standardized API typically used for parallel and/or distributed computing.? This will allow focus upon the database implementation and less on the worries of getting a working distributed system. [http://www.open-mpi.org/]

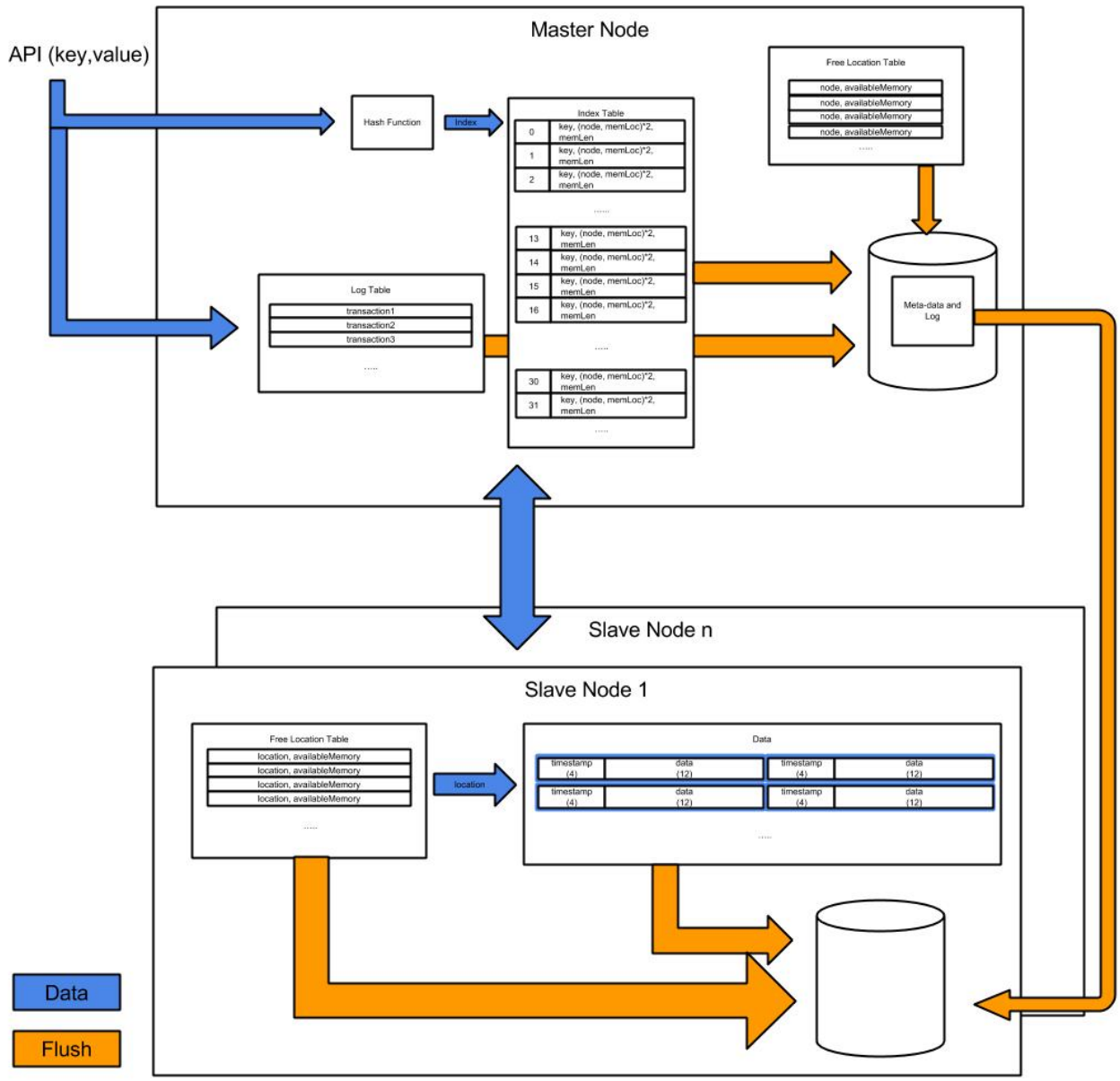

Figure 1: SquirrelDB Component Architecture Diagram