SIMPLEDB SERVER SOFTWARE DESIGN

OVERVIEW

The SimpleDB is made up of 3 main components. The database code is implemented as a shared c library, and the networking/threaded code is implemented in Python. Another shared c library is also used to bind the c functions to Python. The stack looks something like this:

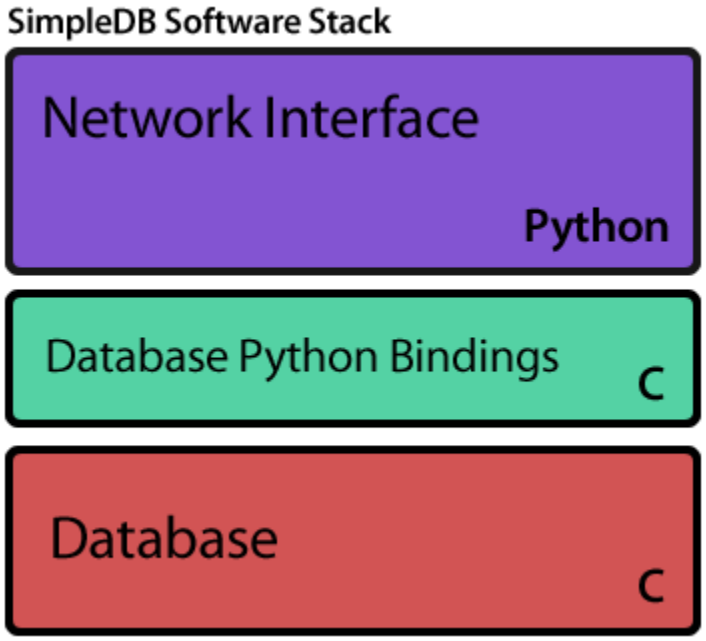

Figure 1 - SimpleDB software stack

All multithreading is done in the network interface which is handled by python threads. Since threading is done at the bytecode level, calls to c libraries can be considered atomic operations. This has some obvious benefits for this particular database server, such as not having to worry about semaphores when handling clients. Even so, the database does not assume that each call is atomic, and implements semaphores that are keyed based on the filename of the database. This way, multiple *servers* could be run without any worry of tromping each other's data.

Bill Casarin, Justin Frenette | Sept 17 2008

SIMPLEDB SERVER SOFTWARE DESIGN

NETWORK INTERFACE

Python threads make the network interface fairly straightforward. Clients are implemented as a class which inherits the threading.Thread object. The main thread sits in a loop and waits for new client connections, when it gets one, it creates a new Client object (which in turn is spawned as a new thread). The Client object holds basic information like client host and port information, the socket, and an initialization flag, which is set on an OHHI message.

The Client thread sits in a loop and waits for requests from the client. If the client disconnects, the loop is exited and the client object and thread are destroyed. When the Client thread gets a request from the client, it looks to see if a function exists in the corresponding Protocol object, and then calls that function.

The Protocol object then goes ahead and calls the corresponding function in the c database library.

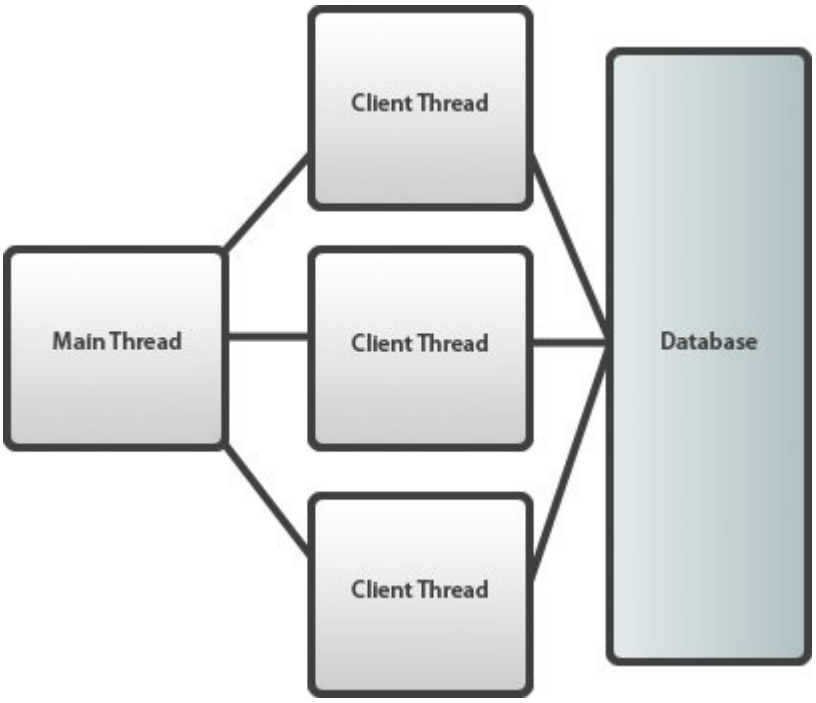

Figure 2 - Networking interface

SIMPLEDB SERVER SOFTWARE DESIGN

DATABASE

The database library is where most of the heavy lifting is done in the server. It was designed with speed and efficient use of space in mind. The database keeps track of 3 files: the database (sdb), the index (sdi), and the orphan file (sdo), which I'll get to later. Data is stored the sdb file. It is in a binary format, and strings are stored in-place with null terminators to signify the end of the string. No size information is stored with the string.

The index file is a binary file which stores offsets to records in the sdb file. For example, an integer at position 0 in the index file would contain an offset to the record with id 1 in the sdb file. An integer at position 4 would contain an offset to the record with id 2 in the sdb file, etc.

Before I explain the orphan file (sdo), I have to explain how inserts and updates work. Database insertions work by appending a new record to the end of the database file and updating the index file. Updates work similarly, if the update doesn't fit in its original position, it will insert the update at the end of the file and update the index to the new position. The old record is then considered to be an *orphan* and is added to the orphan list.

The orphan list is a sorted doubly-linked list which holds the size and offset all of the records that have been orphaned from an update. All updates and inserts look through the orphan list before adding itself to the end of the file. If it finds unused space that it can fit in, it will use it. Anytime the orphan list is changed in memory, it is written to the disk for persistence. The orphan list is only loaded from the disk during database initialization.

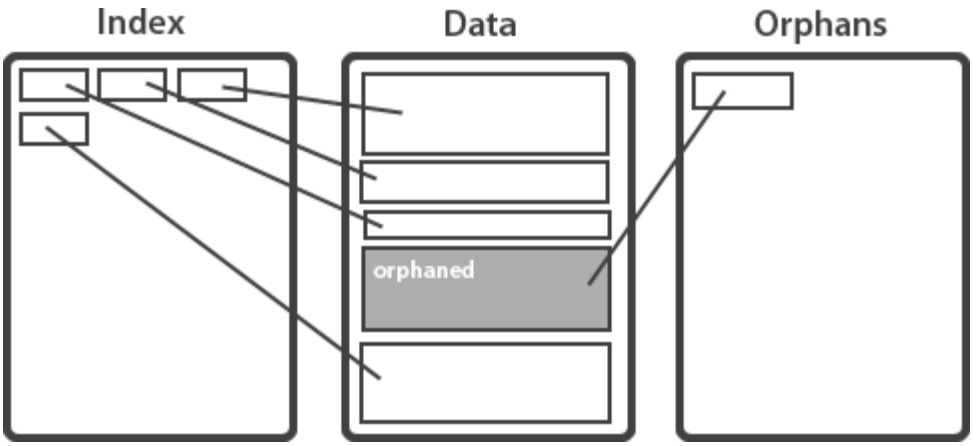

Figure 3 - Database files