Section 8: Key-Value Stores

Frank Austin Nothaft

July 22, 2015

Contents

1 Warmup 2
  1.1 ACID Semantics ......................................................... 2

2 Problems 3
  2.1 Practical ACID .......................................................... 3
  2.2 Simple Key-Value Store ............................................... 3
    2.2.1 Sketch Design ...................................................... 3
    2.2.2 Code Design ........................................................ 4
1 Warmup

1.1 ACID Semantics

In class on Monday, we introduced the concept of ACID semantics. What do each of the letters in the ACID acronym stand for? Can you describe the semantics each letter provides?

A:

C:

I:

D:

Why do we care about ACID semantics? Why would it be difficult to use a database that did not provide ACID semantics?
2 Problems

2.1 Practical ACID

Do you think that all databases provide full ACID semantics? If you think they don’t, why don’t they? What might be the hardest part of ACID semantics to provide?

2.2 Simple Key-Value Store

2.2.1 Sketch Design

We want to implement a simple iterative key-value store that runs on a single, multithreaded machine. In our simple KVS, our keys and values are restricted to be integers. Sketch out what your implementation looks like. In your sketch, show where synchronization is necessary between the threads, and show what synchronization primitives you are using. Use the simple message passing API we introduced last section, and assume that there is one directory thread, and \( n \) key/value storage threads. For this problem, assume that key/value pairs are not replicated.

As a reminder, here is our message passing API:

```c
int msg_send(inbox_t * inbox, void * msg, size_t msg_size);
int msg_recv(inbox_t * inbox, void ** msg);
```
2.2.2 Code Design

Let’s implement the design you just sketched out. Assume that the following functions are defined:

```c
// Returns the number (between 0 and num_nodes) of the subpart of the KVS that
// owns a key.
int get_key_owner(int key, int num_nodes);

// A thread-unsafe map-like data structure that stores integer key-value pairs.
typedef struct int_map;

// Getter/putter functions for the map.
int get(int_map * map, int key);
int set(int_map * map, int key, int value);
```

New structs that you’ve added, if any:

Global variable definitions, if any:
Key getter function:

```c
int get(int key, mailbox_t * my_mailbox) {
```

Key setter function:

```c
int set(int key, int value, mailbox_t * my_mailbox) {
```
Directory message processing function:

void directory_thread() {

}

Key-value storage thread processing function:

void kvnode_thread(int node) {

}