Design Document: adding aliases to the HTTP server

Yujia Li
CruzID: yli302

1. Goals

Implement name mapping to handle aliases. When send PATCH request, update the mapping of old_name to new_name. PUT and GET requests handle aliases, too.

2. Changes in client and server

2.1 Client

2.1.1 PUT & GET

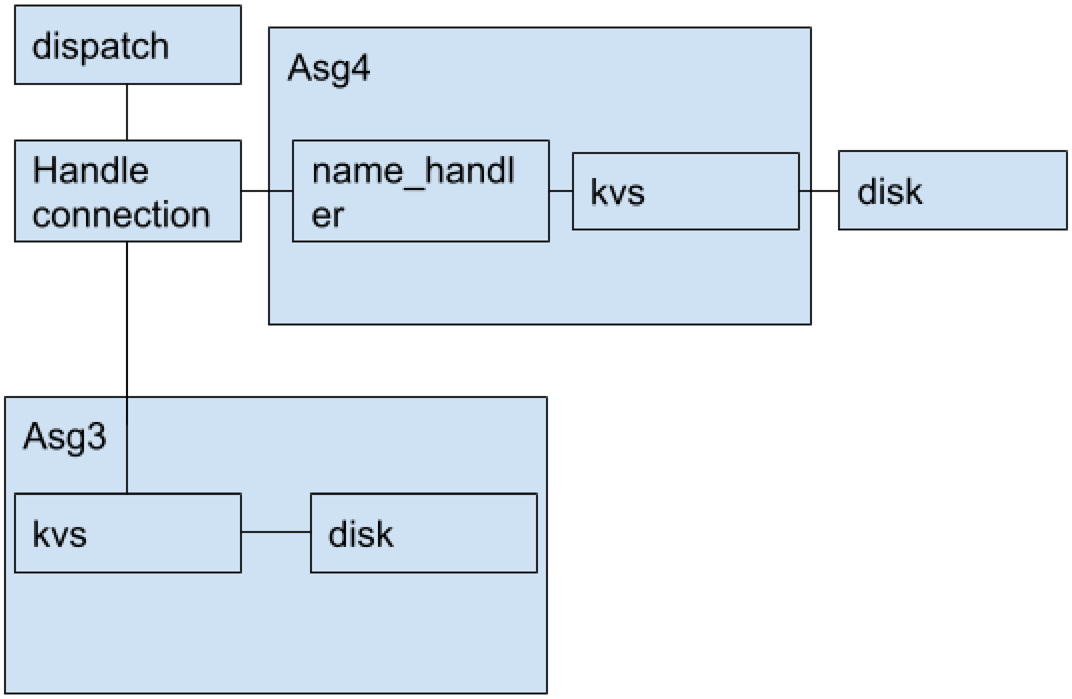
Due to implementation of aliases, client can send httpname without being bound of 40 hexadecimal characters.

2.1.2 PATCH

```
PATCH existing_name HTTP\r\n\r\n
ALIAS existing_name new_name\r\n
```

PATCH request is implemented. The command to send PATCH request is `a:existing_name:new_name.` Client parses command to HTTP header and body, then sends it to server. Server will send response back. Response can only be “200 OK” and “400 Bad Request“ and “404 Not Found”

2.2 Server



2.2.1 PUT & GET

1. Receive request from client and parse (remove leading slash, add null terminator). If length of name is 0 after removing leading slash, send “400 Bad Request” back.
2. Call `name_find_httpname`, and get the corresponding 40 hexadecimal characters httpname. (see 3.1.2)
3. If alias chain is detected, send 508 Loop Detected.
   If httpname doesn’t exist, send 404 Not Found.
4. Process PUT and GET request. No changes.

2.2.2 PATCH

When receive PATCH request, server will:

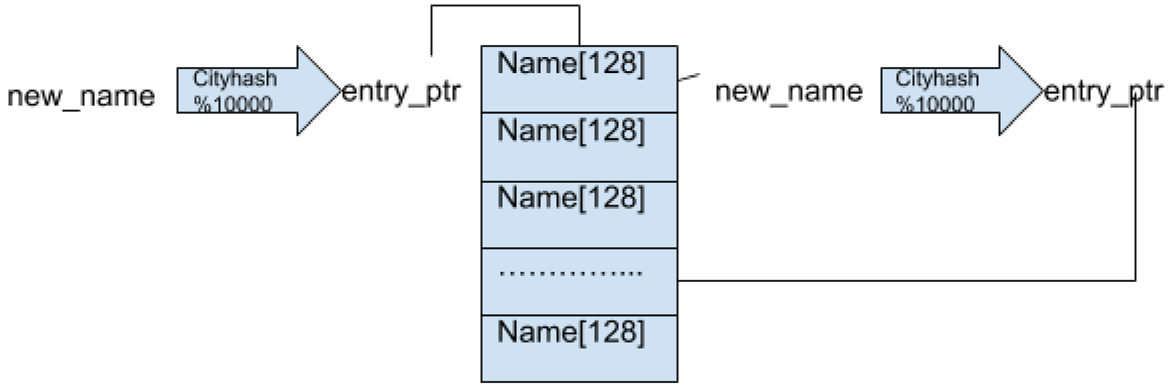
1. Receive request from client and parse (remove leading slash, add null terminator). If length of name is 0 after removing leading slash, send “400 Bad Request” back.
2. Call `name_add`, and update the mapping of existing_name and new_name. (see 3.1.3)
3. If existing_name is not exist in name mapping, and not a valid httpname, send 404 Not Found
   If existing_name + new_name + two null terminator greater than 128, send 400 Bad Request
4. Send response from server to client.

2.2.3 PATCH concurrency

Assuming a name ‘111’ exist in naming map. When run `a:111:222 and a:111:333` as the same time, race condition may happen. Using a semaphore called `write_namemap` to ensure there is only one thread write to kvs_name_mapping. (similar to read and write).

3. Name mapping system design

3.1 Top level design



3.1.1 name mapping entry

Entry will contain name(alias)->name(alias or httpname).
Each name contains ‘\0’ at end. Two name plus two ‘\0’ is maximum 128 byte.
`uint8_t names[128]` contains new_name and existing_name.
New_name is `names[0]` to first null terminator . Existing_name is `names[strlen(names)+1]` to second terminator.

3.1.2 name_lookup

See the diagram above. By passing `new_name` to hash function, `name_lookup` get the `entry_ptr`, and return `existing_name`. See more detail on 4.2.

3.1.3 name_add

Set the new name mapping from `new_name` to `existing_name`. Existing_name must be a 40 hex char httpname or an existing alias. Name cannot be 0 byte. See more detail on 4.2.

3.1.4 name_find_httpname

Call name_lookup until find a valid 40 hexadecimal character httpname or empty entry. See more detail on 4.2.

3.1.5 linear probing

Collision of name entry can happen. Therefore, linear probing is necessary for handle collision.

3.1.5.1 Insert

When collison happen, plus `entry_ptr` by 128(sizeof entry), until find an empty entry. Then insert to this empty entry.

3.1.5.2 Find

After get entry by `entry_ptr`, compare new_name. If different, plus `entry_ptr` by 128, until find an entry contains same new_name, or find an empty one.

3.2 initial name mapping file

Use [-m name_mapping] to specify file name of name mapping file.
If the name mapping file exists, return 0 to notice this name mapping file exists. If the name mapping file doesn't exist, create new one, and return 1 to notice this name mapping file has been created.

To create name mapping file, set 1,280,000 (get by 128 * 10,000) byte as empty entries. See more detail on 4.2.

4. Key-value store data structure & algorithm (for name mapping)

4.1 data structure

`uint8_t names[128]`

Including: `new_name[] + '\0' + existing_name[] + '\0'`

4.2 algorithm

```
// Parse name
char* remove_leading_slash(char* name){
    char* name_without_slash;
    strcat(name, '\0');
    strcpy(name_without_slash, name);
    if name[0] == '/' then
        name = name + 1;
    end
```

```
     return name;
}

// Check whether name is 40 hex characters
bool is_40_hex_httpname(const char* name){
     httpname_Bool = 1;
     if strlen(name)==40 then
          for(size_t i = 0; i < strlen(name); ++i){
               if ((name[i] > '0' && name[i] < '9') ||
                  (name[i] > 'a' && name[i] < 'f') ||
                  (name[i] > 'A' && name[i] < 'F')){}
               else
               httpname_Bool = 0;
               end
          }
     else
          httpname_Bool = 0;
     end
     return httpname_Bool;
}

// initialize name mapping file
int32_t init_name_mapping(){
     uint8_t names[128];
     memset(names, 0, 128);
     offset = 0;
     if (file_sz == 0) {
         for (int i = 0; i < 10000; ++i) {
           pwrite(fd_name_mapping, names, 128, offset);
           offset += 128;
         }
         return 1;
     }
     return 0;
}
```

```
// look up name mapping
// if name mapping exist, return length of existing_name.
// if not exist, return -2.
ssize_t name_lookup(cosnt char* new_name,
                       char* existing_name){
     entry_ptr = Cityhash32(new_name) % 10000;
     char* names;
     // loop for linear probing
     for(;;){
          pread(fd, names, 128, entry_ptr * 128);
          char* name1, name2;
          strcpy(name1, names[0]);
          strcpy(name2, names[strlen(names)+1]);
          if strcmp("", name1) == 0 then
               return -2;
          end
          if strcmp(new_name, name1) == 0 then
               strcpy(existing_name, name2);
               return strlen(name2);
          end
          entry_ptr += 1 * 128;
     }
}

// add new name map from a new_name to an existing_name
// new_name + '\0' + existing_name + '\0' <= 128 byte
// if add successfully, return size of content stored in
// entry.
// if add names greater than 128 byte, return -1.
// if existing name doesn't already exist on the server,
// return -2.
ssize_t name_add(const char* new_name,
                   const char* existing_name){
     char* names;
     if((strlen(new_name)+strlen(existing_name)+2)<=128){
          if(name_lookup(existing_name, nullptr)!=-2
                || is_40_hex_httpname(existing_name)){
                  entry_ptr = Cityhash32(new_name) % 10000;
                  // loop for linear probing
                  for(;;){
                       pread(fd, names, 128, entry_ptr * 128);
                       char* name1;
                       strcpy(name1, names[0]);
                       if strcmp("", name1) == 0 then
                            strcat(names, new_name);
                            strcat(names + strlen(new_name)+1,
                                    existing_name);
                         return pwrite(names,128,entry_ptr*128);
                       end
                       if strcmp(new_name, name1) == 0 then
                            memset(names, 0, 128);
                            strcat(names, new_name);
                            strcat(names, existing_name);
                         return pwrite(names,128,entry_ptr*128);
                       end
                       entry_ptr += 1 * 128;
                  }
             }
             else{
                  return -2;
             }
        }
        else{
             return -1;
        }
   }
```

```
// look up name mapping until find valid httpname or empty
// entry.
// A alias chain can occur. In This situation alias will
// always map to another alias and never map to a valid
// httpname. Use a vector<string> to check alias chain.
// if find alias chain, return -1.
// if find valid httpname, return its length.
// if find empty entry, return -2.
ssize_t name_find_httpname(const char* new_name,
                           char* existing_name,
                           vector<string> nameList){
    // new_name is a valid httpname
    if(is_40_hex_httpname(new_name)){
        strcpy(existing_name, new_name);
        return strlen(existing_name);
    }

    // entry is empty
    if(name_lookup(new_name, existing_name)== -2){
        return -2;
    }
    // existing_name is a valid httpname
    if(is_40_hex_httpname(existing_name)){
        return strlen(existing_name);
    }
    // existing_name is alias.
    string s(existing_name);
    // an alias chain is detected
    if(find(nameList.begin(),nameList.end(),s)!=
       nameList.end()){
        return -1;
    }
    nameList.push_back(s);
    char* nextname;
    strcpy(nextname, existing_name);
   return name_find_httpname(nextname, existing_name,
                            nameList);
}
```