Assignment 6 Part 2 Memo

Kate Rupar and Mark Thekkethala

Introduction

This memo introduces our serialization support class. In the following sections we will describe the Message classes and serialization/deserialization processes.

Message and MessageKind

The message class serves as an interface for the different kinds of messages sent between systems in the Network. It was provided in the spec and has three fields: MessageKind, sender Node index and target Node index. In our design, the sender and target indexes refer to positions in a list of addresses and ports in a directory. MessageKind is an enum class with ten types. In our implementation we only focus on the four types required to implement according to the spec: Ack, Status, Register and Directory.

Ack

The Ack class is very basic. It inherits from Message and contains no extra fields. We send Ack Messages to signal that a Status has been received.

Status

The Status class has an extra String* field, msg. We use the Status Message to send messages between Nodes and from Nodes to the server.

Register

We modified the given Register class to utilize a String* address instead of sockaddr_in client, since we only need the client IP in order to add to the directory's IP address list. Register also contains a port. Only Nodes can send Register messages to the Server, not vice versa. We assumed that register methods can also only be sent once to the server per Node, ie once it is registered it cannot re-register. Register Messages are used to add a Node to a Server's directory, and receiving a message triggers a request for a Directory Message to be sent from the Server to the Node.

Directory

The Directory class has a number of nodes, list of ports, and list of IP addresses. Directory Messages are sent only from a Server to a Node after a Node is added to the Network so Nodes can update their Directories to include the new Nodes information.

Serialization

The Message class has a virtual method serialize which all of the above mentioned classes inherit. In each Message class, the fields are converted into characters and appended, with the ‘?’ character separating the fields. Therefore, serialize returns a String*. We chose ‘?’ to be a forbidden character in a Status message. In the case of Directory, we include all of the elements of the port and address lists in the serialized string. The first character of the serialized Message represents the message kind: 1-Register, 2-Ack, 3-Status, 4-Directory. For example, a serialized Message “3?0?1?hello” would refer to a Status message from Node 0 to 1 with the message “hello”.

Deserialization

For deserialization, we created constructors for each Message type which take in a char* (the serialized string) and return a Message. In our node.h handle_packet() method, we look at the first character of the serialized string to determine the MessageKind and call the appropriate constructor.

Use Cases

The following are use cases for serialization in pseudocode:

```
//Determining if we can serialize and deserialize to get back an object with
//the same fields as the original
Status* s = new Status(0, 1, new String("hello"));
Status* f = new Status(s->serialize()->cstr_);
f->serialize()->cstr_ == s->serialize()->cstr_

Register* r = new Register(0, 1, 8080, new String("127.0.0.1"));
Register* e = new Register(r->serialize()->cstr_);
r->serialize()->cstr_ == e->serialize()->cstr_

Ack* a = new Ack(0, 1);
Ack* g = new Ack(a->serialize()->cstr_);
a->serialize()->cstr_ == g->serialize()->cstr_
```

```
size_t* ports = new size_t[3];
ports[0] = 1;
ports[1] = 1;
ports[2] = 1;
String** add = new String*[3];
add[0] = new String("127.0.0.1");
add[1] = new String("127.0.0.1");
add[2] = new String("127.0.0.1");
Directory* d = new Directory(0, 1, 3, ports, add);
Directory* c = new Directory(d->serialize()->cstr_);
d->serialize()->cstr_ == c->serialize()->cstr_
```

Future Applications

Although in this project we only serialized Statuses which carry String messages, our design can easily be reconfigured so that Status contains a SoR type or array of SoR type. We can utilize the same logic as in Dictionary to serialize each field and send a Column or even entire DataFrame across Nodes.