Mini Project 3

jglr
emja
krbh
thhk
(Group AD)

November 16th 2020

Part A

1 How to run the program

Setup

Run
`javac *.java`
from the root directory to compile.

To run Node.java

Run
`java Node.java <localport> <IP> <port>`
Where <localport> is the local port that other nodes should use to connect to the node.
<IP> and <port> are optional parameters, that define the IP and port of another node.
To start a node without connecting to another node, you could for example run
`java Node.java 1337`
To start a node that connects to this node, you could then for example run
`java Node.java 1338 10.26.55.65 1337`

To run Put.java

Run
`java Put.java <IP> <port> <key> <value>`
Where <IP> and <port> specify the node to put to.
<key> is an integer that specifies the key of the key-value pair,
and <value> is a string that is the value.
You can for example run
`java Put.java 10.26.55.65 1337 3 sample`

To run Get.java

Run
`java Get.java <IP> <port> <key>`
Where <IP> and <port> specify a node to get from.
<key> specifies the key of the key-value pair that you want.
If you wanted to get the value from the example that we used in the previous section, run
`java Get.java 10.26.55.65 1337 3`
Keep in mind that the IP and port just needs to point to a node that is connected to
the same net work as the node with the value. You don't necessarily need to point to the exact node.

2

Is your system... Why/Why not?

2.1 A publish publish/subscribe system?

No, you do not subscribe to a service. A node does not subscribe to a service.

2.2. A message queue?

No. Nodes are not notified when a new Put has been added. Only the node that has received the Put knows this.

2.3. A structured P2P system?

Yes. It follows a ring-structure.

2.4. A unstructured P2P system?

No, as it is structured.

2.5. A distributed set implementation?

No. Put-objects are key-value pairs, not just values.

2.6. A distributed hash table implementation?

Yes. When you make a Get for a specific key, the system will be searched for that specific key and its value.

3

What are the average-case, best-case, and worst-case space consumed at each Node?

The space scales linearly with the number of Puts, that each Node has. However, the space is O(1) for the number of Nodes in the system. In the best case, the Node is the only Node in the system, in which case, it does not have to remember any other nodes. In the average and worst case, there are 3 or more Nodes in the system, such that a Node need to remember 2 other Nodes.

4

What are the average-case, best-case and worst-case number and size of messages being sent as a result of...

For message sizes view appendix.

4.1. A PUT message from a client.

A PUT message will in all cases require 2 messages, including the original PUT message. One message receiving the original PUT message, and one more sending it on to another Node.

4.2. A successful GET message from a client.

Best case will be that the Node receiving the GET message first, contains what the GET message is looking for, in this case 2 messages are send, firstly the GET is send to the Node, then the Node send back the answer.

Worst case will be if the Node twice before the Node originally receiving the GET message is the is the Node that originally received the PUT message that is being requested by the GET message. This would require N-1 messages to be send. 1 message to the Node originally receiving and N-2 messages to reach around to the first Node containing the requested PUT message.

4.3. An unsuccessful GET message from a client.

A unsuccessful GET message will in all cases require N messages, including the original GET message. One message receiving the original GET message, and N-1 messages sending all the way around the ring structure, to the Node before the Node that originally received the Get message.

Part B

1

Briefly explain your solution to Part B and why it works.

Every node has a reference to the two nodes ahead in the ring structure. This design entails that if one node crashes, then the ring structure is only partially disconnected and furthermore brings the information needed to reconnect the ring, given that no subsequent crashes occur.

2

What are the average-case, best-case and worst-case number and size of messages being sent as a result of ..?

For message sizes view appendix.

2.1 A PUT message from a client

A PUT message will in all cases require 2 messages, including the original PUT message. One message receiving the original PUT message, and one more sending it on to another Node.

2.2 A successful GET message from a client (that is, a value is found and sent back.)

Best case will be that the Node receiving the GET message first, contains what the GET message is looking for, in this case 2 messages are send, firstly the GET is send to the Node, then the Node send back the answer.

Worst case will be if the Node twice before the Node originally receiving the GET message is the Node that originally received the PUT message that is being requested by the GET message. This would require N-1 messages to be send. 1 message to the Node originally receiving and N-2 messages to reach around to the first Node containing the requested PUT message.

2.3 An unsuccessful GET message from a client (that is, no value is found.)

A unsuccessful GET message will in all cases require N messages, including the original GET message. One message receiving the original GET message, and N-1 messages sending all the way around the ring structure, to the Node before the Node that originally received the Get message.

3

Write a paragraph or two suggesting improvements to scalability that does not compromise your one-node-resiliency.

It would be possible to change the get-method of the Node-class to start trying to send to its secondToNode, and then if this fails send to it toNode. Implementing this would require that a Node check whether it toNode is the one that originally received the get-object, as to stop sending the get-object around more than necessary. However it would half the amount of messages needed, as it would skipping every other node, allowing for more nodes to be added to the network before performance deteriorates to much.

4

If your solution provides exactly one-node-resiliency, write a paragraph or two suggesting methods your one-node-resiliency can be expanded to n-node resiliency, assuming your network has time to reconfigure itself in between node failures.

If a Node, x_1, gets a connection exception when trying to connect to its *toNode*, it will note this internally. When x_1 is done handling the request where it detected this error, it will send to its *secondToNode* (x_2) a Repair-object containing x_1's address. x_2 adds to this *Repair*-object its own address as well as its *secondToNode*'s (x_3) address. With this information the Repair object is finalized, and will be send all the way around the ring structure. When a Node having x_1 as its toNode address receives this Repair-object it will set its secondToNode to x_2's address, and send the Repair-object to x_1. x_1 will then set it toNode to x_2, and secondToNode to x_3, the Repair-object is not send further and the ring structure is now fixed.

Appendix

1. Object sizes

Object	*Condition*	*Size* (bytes)
ADDRESS	-	40
PUT[1]	-	(16 + 2x) + 29 + p
GET	Initial send	48
GET	Subsequent sends	88
CONNECT	When step is 0	64
CONNECT	When step is 1	88
CONNECT	When step is 2	64

[1] where x is the string length, and p is object padding