MINI PROJECT 3

Awis - Afin - Cnbl - Djam

PART A

1. Is your system...

A publish/subscribe system?

No. Nodes do not subscribe to anyone, but just listens. Instead a Node can have responsibility to send a message on to another node.

A message Queue?

No. Since no queues or centralized servers are involved.

A structured P2P system?

No. The Get call takes O(N) time in the worst case, since nodes are connected in a long line, and not in a pastry format.

An unstructured P2P system?

Yes. The system is a P2P system where nodes functions as peers. The System is unstructured since the Get function takes O(N) in the worst case.

A distributed set implementation?

A distributed hash table implementation?

No. But by doing so one could achieve a structured P2P system.

2. *What is the average-case, best-case, and worst-case space consumed at each Node?*

NOTE: E is the amount of Puts(messages), N is the amount of nodes.

Worst Case for each Node is E where all messages are on one single node.
Best Case for each Node is 0 where all messages are on other nodes.
Average Case for each Node is E/N such that all nodes have an equal amount of the messages

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

A PUT message from a client

The Put message is called to a single node and therefore the amount of messages are constant for all three cases.

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

Worst case O(N), average case O(N) or N/2 and best case is 1.

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

For all cases the result is infinite as the GET request will initiate an in infinite loop in the network.

4. Based on 2 and 3, write a paragraph or two on the current scalability of your system.

The scalability is quite poor right now. In case of failure the system is almost unusable and for a few thousand peers this is quite unuseful. The problem, as even with native running applications, is that when working with a linked list one minor fault results in a devastating fault of the data structure, distributed or not.

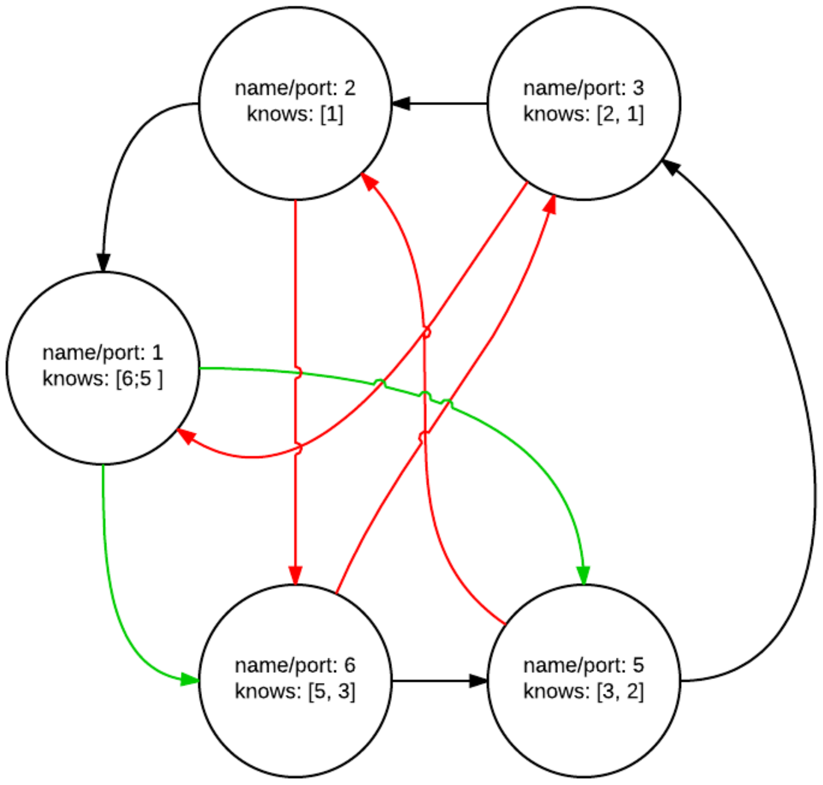
5. Based on 2, 3 and 4, give suggestions for improving the scalability of your system.

By implementing a structured P2P system such as the pastry structure one could improve the Get time and space consumption at the nodes. A Timeout functionality could make the system failsafe such that a constant and not infinite time consumption in case of a unsuccessful GET. The Heartbeat functionality could improve the system in the case that a node leaves the system.

Part B

1. Briefly explain your solution and why it works.

The modified solution to support resilience of failing nodes has been implemented through an increased awareness per node. Each node now knows about not only its prior node, but also the node before that. This is illustrated in the figure below; black arrows represent primary links between nodes, red representing secondary and green arrows represent the special case scenario which is connecting the first node with the last node, effectively implementing a ring.



To obtain this network each node must exchange information when a node is initially added, as well as continuously maintain the structure:

When a node is added it is informed about its prior node (primary child) and communicates with it to set its alternative node (secondary child). This creates a one directional linked list. To fulfill the specification of a P2P system, *information in the network may be retrieved from any node*, a relationship between the first and last node must be obtained. This is done when a node is added to the system, initiating a ROOT message containing the communication address of the last node. The ROOT message is continuously passed to nodes children until it reaches the ROOT node, which updates its children in accordance to the message.

Note that the root node is detected when the nodes identity number is lower than its children.

To maintain the structure each node has a “heartbeat” that pulses to its two known children to test whether they are still alive or not. This occurs at an interval of 3 seconds. If an error is detected via a heartbeat the nodes references must be reestablished which is done by adapting the references of the other child..

2. & 3. Space consumption and number and size of messages being sent.

Space consumption per node has, in comparison to the non-resilient version, increased by a constant. This constant is a field that describes the existence of another node - now two, which prior was only one.

In regards to the amount of messages that are sent when sending PUT and GET messages, the count remains the same. The size of the messages has not increased, as the language of communication has merely been extended to support the “SECONDARY” and “ROOT” requests

It is though noteworthy that the overall bandwidth used in this network design has significantly increased due to the large amount of *check, test and reconfiguration messages* periodically sent.

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

The main challenge of the implementation described is in the case of cluster failure where several nodes closely related are lost from the network simultaneously. If two nodes that are logically nabors (children) fail, the network becomes unrecoverable. This has a high probability when looking at what challenges scalability would introduce. To strengthen the design two improvements come to mind:

1. A mechanism that supports the lookup of nodes in a range in the case of cluster failures, as seen in structured P2P systems.
2. Bi-directional messaging. This would make partial or whole network recoveries possible even if large sections where lost, but would double the bandwidth consumed by the network.

5. *~~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.~~*