Tomas Hasselquist, 920811-1097
David Gardtman, 920228-4718
Group: 22

Report Lab 1 - Reliable & Ordered Multicast

1. Introduction

This report describes the algorithms that can be used in order to have a reliable and ordered multicast. To summarize our solution makes use of a sequencer which every process have to send their messages to so that it may add a sequence number to it before broadcasting so that all correct processes deliver the same set of messages in the same order.

The basic algorithm is quite simple in itself. It uses the reliable multicast algorithm code provided in lecture 2. It utilizes a dynamic client-server system where a sequencer is selected to handle ordering of messages. It starts of by selecting a sequencer by putting the node with the lowest id as sequencer. When a message is created, it receives a local sequence number from the node and then gets sent to the sequencer. This way the sequencer can add a global sequence number to make sure all messages can be delivered in the same order.

To keep track of this we have implemented four different message bags to hold messages depending on what state they are in. Firstly we have the normal message bag (msgBag) which is used to store received messages that needs to be delivered. This could contain the next expected message to be delivered or perhaps some message that still requires a few earlier messages to be delivered first.

The second bag is the history bag (delivery). This bag is used to keep track of all delivered messages. This is to make sure we do not deliver a message twice. Although it is not necessary since we can simply keep track of the expected sequence number we felt it would be a nice implementation for possible future changes.

The third bag is the bag to keep track of a node’s own messages that has not been delivered (notDelivered). This bag is used as a safety net in case the sequencer node crashes. This way the original creator of the messages can resend any messages that got lost by the crashing sequencer.

Finally we have the fourth bag which is the (noGlobalSequence). This bag works the same way as the first message bag (msgBag) except it is used by the sequencer in case the node received a message from a specific node where the local sequence number is not the expected sequence number from that node.

Tomas Hasselquist, 920811-1097
David Gardtman, 920228-4718
Group: 22

2. Answers

2.1 Integrity

The solution satisfies integrity since every correct process will only deliver a message m once. This is done by using a sequencer that adds a global timestamp to the message and this global timestamp is unique to each message. Therefore, when a node receives a message from the sequencer and rebroadcast it to all other neighbours, they will not add the message again if they receive it, since they can see that they have already delivered a global timestamp of that value.

Furthermore, the sequencer does not give the same message a global timestamp twice which means it cannot fool the other nodes to deliver the same message twice. This is established by the sequencer since the received messages that need a global sequence number has the local sequence number in the message. This way the sequencer will add the global sequence number to only the expected messages.

If processes crash integrity will still hold. If the sequencer crashes after it has sent the message with the global timestamp the original process will receive this message and remove it from its notDelivered bag and therefore not ask the new sequencer to give a global timestamp again. Furthermore, all nodes update the local sequence numbers as well as the global sequence number of the received messages to make sure they can take over the role as sequencer and start checking for the next expected message from each node.

2.2 Validity

Our solution also satisfies validity since a broadcasted message will always arrive because we assume that the connections can not fail. To ensure that the message reaches all processes each process re-broadcasts the messages with global sequence numbers once to all neighbours except the sender so that the message might propagate.

If the sequencer would crash before a message received a global sequence number the process that asked for a global sequence number will simply wait until a new sequencer is appointed after which it will resend all not delivered messages. We can therefore ensure that all correct processes will resend their messages that have not been given a global sequence number and all correct sequencers will give these messages numbers and broadcast the messages to the whole system. This means that all correct processes will eventually deliver the same set of messages broadcasted by the sequencer.

Tomas Hasselquist, 920811-1097
David Gardtman, 920228-4718
Group: 22

2.3 Agreement

Since the sequencer needs to have added a global timestamp for the other processes to actually deliver the message we put the burden of agreement on a single process. The other processes simply have to receive the broadcasts made by the sequencer to ensure that all processes agree. Because of our re-broadcast algorithm we can make sure that all messages eventually arrive in all correct processes. Because of the global timestamp they will also be delivered in the correct order which means we can be sure that agreement is satisfied.

This still holds even if a node crashes since all messages are re-broadcasted the first time they receive the message. For example if the sequencer only messages one node about a global message before it crashes this node will propagate the message to the rest of the system so that agreement will still hold.

2.4 Ordering requirement

Each process has a logical clock which tick at events, this provides local ordering. This local timestamp is added to the message when sending it to the sequencer. To ensure ***causal ordering*** the sequencer will wait for earlier local timestamps from a process if it has not received them before adding a global timestamp and broadcasting. These messages are added to the noGlobalSequence bag which is checked each time a new message arrives to the sequencer. This means that all messages from a certain process will be delivered in the same order on all processes, though they may be interleaved between other processes' messages. In this way our solution provides causal ordering.

Global timestamps also provide ***total ordering***. Each message broadcasted by the sequencer will be received by all other processes. If a broadcast arrives to a process with a higher global timestamp than it expected (i.e. one more than the previous global timestamp it received) it will add this message to its msgBag and wait for the expected message before delivering the message. Once the expected message has been received and delivered, the process goes through its msgBag to see if it has the next expected message. This is done iteratively until it has delivered all incrementing expected messages in the msgBag. Since all processes follow this logic total order will be achieved since all processes deliver messages in exactly the same order.

2.5 Crashing processes

Since Mcui sends information about crashed processes to all other processes it is quite simple to deal with crashed processes. Every process keeps a list of booleans the size of number of processes called active. When a message arrives saying a certain peer is down the corresponding boolean value in the list is set to false.

If the sequencer dies all processes iterate over the list until we find the first (lowest id) process that is still active and set this new process to be the new sequencer. If a node has

not realized (computed) that he is the new sequencer and receives a message that needs to be sequenced he will add this message to his noGlobalSequence bag in case he will become the sequencer soon. Once this occurs, the new sequencer will go through this bag as usual. If this node crashes before becoming leader the node that sent the message will get information about the crash, look through his notDelivered bag and send the same message to the next node that is leader.

Since the messages are not removed from notDelivered until they are actually delivered we can ensure that no messages are lost in case of several sequencer failures. This helps ensure validity.