Liégeois Arthur et Bequet--Ermoy Charles

Breaking the scalability barrier of causal broadcast for large and dynamic systems

by Brice Nédelec, Pascal Molli and Achour Mostéfaoui

Problem statement

In a network of processes where messages are sent, it is sometimes needed to keep the order of messages as they may be related. In social media applications, a message should appear before the responses to that message do.

Causal Broadcast is very important in distributed applications as it aims to make every process of a network deliver messages in the causal order. It relies on the reliable broadcast, which ensures that every message broadcasted will be delivered by every correct process exactly once, and it follows the “happen before” relationship.

But causal broadcast is very expensive in large environments where each process forwards every message it delivers to its neighbors in order to reach everyone in the network. This is called forwarding or gossiping.

State of the art

One possible approach is to make every message piggyback a vector clock to ensure causal order. This way you can know what messages to wait for before delivering another. But in dynamic systems, the size of that vector grows linearly with the number of process that ever broadcasted a message in the system as well as the delivery execution time which will be in O(N.W) with N being the number of processes that ever broadcasted a message and W the number of messages awaiting delivery.

This approach is as inefficient as it is expensive.

There is another approach which uses only FIFO communication means. This means a process will deliver the messages received from another process in the same order they were sent by that said process. This ensures causal order as well as being efficient and lightweight as there is no delivery execution time, messages can be delivered instantly and there is no message overhead as you don't need to piggyback information to ensure causal order. But in dynamic systems we can add and remove communication means as we go. Using this approach in a dynamic system will result in causal order violations. If a process gets a new neighbor and starts forwarding messages to it, there is no guarantee that this new neighbor delivered all the messages preceding these ones.

So state-of-the-art protocols are either overcostly or cannot work in dynamic systems. This paper presents a new approach based on the FIFO approach but with some enhancements that allow it to resist the changes due to the dynamic environment.

PC-Broadcast

PC-Broadcast stands for Preventive Causal Broadcast, as it aims to prevent causal order violations from happening instead of correcting them afterward. Let's consider a network where each node knows only a subset of the network. We suppose that this network is unpartitioned as it is otherwise impossible to broadcast a message in the whole network.

This paper introduces the notion of unsafe links. When a new mean of communication is added between two nodes (that we call A and B in this example), there may have some messages travelling between those two (by another path) that have not arrived yet. That is why we cannot use this new shorter path yet and we need to label it a "unsafe".

To make the new path "safe", we need to make sure that all the messages forwarded by A using the old path are received by B. That is why we initiate a "ping phase", in which A sends a ping to B using the old path and stops forwarding the incoming messages to B until the pong is received (these messages are put in a buffer). When the ping is received, B is sure that it is the last message send by A (because of the FIFO nature of the communication means) and it can send the pong back. When A receives it, his buffer is sent to B (in the same order as it was filled) using the new path, this ensures that no message is lost and that the FIFO order is respected.

Performances

The performances of the PC-Broadcast are the same as the FIFO-Broadcast, but it alsos works in dynamic systems, where means of communications between processes can be added or removed.

First of all, this broadcast does not need to add any information on the messages it sends, contrary to to vector clock approach.
Secondly the messages received to not need to wait before being delivered as the system ensures at all time that a message received can be delivered immediately.
Lastly the local space consumption is linearly growing with the number of processes, as each node needs to store only the last message sent to each node.

Future works

This paper introduces some optimizations to the already existing broadcasts but it can still be improved. One of the main problems is the forced order on the messages, when two unrelated messages are sent, we force an order between them. One way to solve this is by including a vector clock with the messages but that would defeat the whole purpose.