Computing Methodologies - Distributed Computing

Tom Robson - hzwr87

1 The Beta Synchroniser

1.1 How the Beta Synchroniser works

Synchroniser Beta begins with an initialisation phase, in which a leader is chosen in the network and a spanning tree is constructed with this leader node as the root. For all the other nodes, in each round, when all the messages sent by a node have arrived at their destinations, the node determines that it is in the safe state. Once it has received safe messages from its children, it sends a safe message to its parent. These messages work up the tree, following a procedure known as a convergecast. When the leader receives safe messages from its children, it broadcasts it to all the nodes in the tree, signaling that they can now begin a new round.

1.2 Comparison of Alpha and Beta

When performing a comparison on a graph $G = (n, m)$, with complexities expressed per round, it is clear that Synchroniser Alpha is superior in terms of time complexity($O(1)$) but inferior in terms of message complexity($O(m) = O(n^2)$), while synchroniser Beta is superior to Alpha in terms of message complexity ($O(n)$) but inferior in terms of time complexity ($O(n)$). This difference is due to the fact that Alpha is optimised for time complexity, as all communication is only between neighbours, but involves a large amount of messages, whereas Beta is optimised for message complexity, as it sends only once message along each edge, but messages must be sent from the leaves of the spanning tree to the root, taking more time.

2 The Gamma Synchroniser

The Gamma Synchroniser is essentially a combination of the Alpha and Beta Synchronisers that is efficient in terms both time and message complexity. Like Beta, it has an initialisation phase, in which the network is partitioned into clusters with a small diameter. In each of these clusters, a leader is selected and a Breadth-First Search Tree, or intracluster tree, is calculated with this leader as its root. For every pair of neighbouring clusters, one edge is chosen as the intercluster edge, along which all communication between the clusters will be carried out.

The Gamma Synchroniser then proceeds in two phases. In phase one, the Beta Synchroniser is applied to every cluster. As soon as each leader knows that its cluster is safe, it reports this fact to all of the nodes in its own cluster as well as the leaders of the other clusters. Synchroniser Alpha is then applied to the set of cluster leaders, using the intercluster edges, as they wait for all neighbouring cluster to confirm that they are safe before starting the next round.

When calculating the time complexity of Gamma, we say the depth of each intracluster tree is at most k. In each of these trees, there are two sets of convergecast messages and two sets of broadcast messages. This has a worst-case time complexity of 4k. There is an additional timestep required to send messages over the intercluster edges, so the time complexity of Gamma is $O(4k + 1)$, which can b simplified to $O(k)$.

When considering message complexity, we note that 4 messages are sent over all intracluster tree edges. The number of intracluster tree edges must be less that n, so the message complexity of this part is$O(4n)$, or just $O(n)$. We must then consider the number of messages sent over the intercluster links, denoted here as m_c. Therefore, the overall message complexity of Gamma is $O(|n + m_c|)$.

The time and message complexities of Gamma are dependent on the number of clusters and their size. Therefore, both Alpha and Beta can be seen as special cases of Gamma, as if the whole graph was spanned by one cluster, it would be an instance Beta, and if each cluster consisted of only 1 node then it would be an instance of Alpha.

3 DFSr Algorithm

3.1 Time Complexity of DFSr

On a graph with n vertices and m edges, the time complexity of DFSr is $O(m)$. This is informed by the fact that all non-root processors can only send a message when they have received a message, and can only send one message to one neighbour at a time. This means that there can only be one message being sent in the system at any one time. As at most two messages pass along each edge in an execution of this algorithm, if we assume that the maximum message delay is 1, then the worst case time complexity is $O(2m)$, simplified to $O(m)$.

3.2 Applying Alpha to DFSr

When applying the Alpha Synchroniser, we introduce an message overhead of $O(m)$ messages per round. The DFSr algorithm has a message complexity of $O(m)$, as it sends one message per round and there are $O(m)$ rounds in an execution of the algorithm. Therefore the message complexity is $O(m^2)$. The Alpha Synchroniser introduces an time overhead of $O(1)$ timesteps per round. Therefore the time complexity of DFSr remains $O(m)$.

3.3 Applying Beta to DFSr

When applying the Beta Synchroniser, we introduce an message overhead of $O(n)$ messages per round, as in the tree structure the worst case is that there are $n - 1$ edges, with two messages needing to travel along each. The DFSr algorithm has a message complexity of $O(m)$, as it sends one message per round and there are $O(m)$ rounds in an execution of the algorithm. Therefore the message complexity when using the Beta synchroniser is $O(mn)$. The Beta Synchroniser introduces an time overhead of $O(n)$ timesteps per round, as in the worst case the tree is in fact a path and $2(n-1)$ timesteps are needed per round to enable Beta to function. Therefore the time complexity of DFSr now becomes $O(mn)$.

4 The Consensus Algorithm

All 3 of the synchronisers fail to solve the consensus problem in an asynchronous system in the presence of crash failures. This is due to the fact that they require all processors to send safe messages before they can begin a new round.

If Alpha is applied to this algorithm, and one of the processors crashes, then it will not have sent safe messages to all of its neighbours. The neighbours require a safe message from this processor to advance to the next round, and if this is never sent, then they will remain stuck in the round that the processor crashed in. The neighbours of these processors will have moved onto the next round and sent out their information, but they will then get stuck as they do not receive a safe message. This will propagate out until the whole algorithm is stuck.

If Beta is used a similar problem will occur, but it will not take time to propagate to the whole algorithm. If a processor fails, then it will not send a safe message, which will not propagate up to the leader, which in turn will then not be able to commence the next round.

As Gamma is effectively a combination of Alpha and Beta, and neither of these can withstand crash failures, then it is clear that Gamma will also be unable to solve the consensus problem.

More generally, in an asynchronous system where 1 or more of the processors can crash there is no algorithm that solves the consensus problem, as in an asynchronous system we cannot tell if a processor has crashed or if it is just slow.

References

[1] James Aspnes. Notes on Theory of Distributed Systems CPSC 465/565: Spring. 2016.

[2] Hagit Attiya and Jennifer Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics.* John Wiley & Sons, 2004.

[3] Baruch Awerbuch. Complexity of network synchronization. *Journal of the ACM (JACM)*, 32(4):804–823, 1985.

[4] Fabian Kuhn. Chapter 10 - synchronisers. 2011.