Reliable Distributed System Approaches

Seminar of Distributed Computing

DCG ETH Zürich

Manuel Graber (graberma@student.ethz.ch)

October 2003

1 Summaries

1.1 Summary of Virtual Synchrony and ISIS

The paper [isis] describes the Process Group approach and an execution model called Virtual Synchrony. Both of these concepts are used in a group communication middleware called ISIS, which has been developed by the author of the paper. ISIS offers several tools that help to build distributed applications using the Process Group approach. These tools are described in detail. The ISIS software has been commercialized and used for several important applications. The conclusion of the paper is that without a helping middleware the development of distributed application is too difficult and complicated for the "usual" application programmer. ISIS offers strong semantics in group membership, communication and synchronization. This allows a programmer which is not familiar with all the problems of distributed systems to develop correct applications.

1.2 Summary of Spinglass

The first part of this paper [spinglass] is an analysis of the conventional systems (like ISIS) that offer group communication. The conclusion is that conventional systems do not scale. This is because the systems use assumptions that are true in small networks, but in large networks, these assumptions may be violated. So the authors decided to develop the Spinglass tools, which offer support for developing scalable distributed applications. This was realized using epidemic-style algorithms (gossip algorithms). The result is a system that offers good scalability but has different, probabilistic, reliability guarantees than the Virtual Synchrony approach described in [isis].

2 Analysis

An achievement of ISIS is that it makes it possible to develop distributed systems following an easy programming model. The programmer does not have to care about all the nasty things that can happen when trying to make a distributed system reliable. He can just use the provided tools and focus on developing the application logic. The ISIS system works properly as several applications showed, but the model is about 20 years old. This means the important changes of the last years in distributed systems are not considered. In very dynamic, mobile systems the assumptions that were made for ISIS may be violated. The same holds for large networks. The result is that the system neither performs well in dynamic settings nor scales in large networks. This is mostly because the developers did not care much about partitions. Only the largest part remains operational, all other nodes are considered to have failed. If a node is considered to have failed, it cannot recover. It has to join like a completely new node. This has a big impact on performance. An improvement of the ISIS toolkit would be to allow rejoin of a node that was incorrectly suspected to have failed. Or more general, some sensible handling of partitions, because a node that is incorrectly suspected to have failed can be considered as a partition of the group into two parts.

The Spinglass approach offers in contrary to the ISIS approach only probabilistic guarantees. Concerning this point the ISIS system offering "hard" guarantees is better.

The Spinglass approach does not have this scalability problem, but there are some other things to consider:

- The ISIS System was developed about 20 years ago. And consequently all the assumptions (delays, global time not possible, ...) that are made in the ISIS system are old, but they are also used in Spinglass. The question is: Do these assumptions still hold nowadays? For example: Are the delays when swapping in/out a process still a problem? What about global time when we use a radio time signal that can be received at all nodes? I think this depends largely on how precisely the nodes need to be synchronized. Sending a radio signal that can be received by all nodes is certainly not a problem today. Synchronizing the local clocks to this radio signal is possible too. But the main problem is that we need a kind of real-time guarantees from the operating system concerning swapping and interrupts. If for example a radio time signal is detected from the receiving device it must be delivered to the application after some fixed period of time. I doubt that this is possible specially concerning swaps. Perhaps probabilistic guarantees could be considered, but this would change the characteristics of such a system. But what if we find out after some detailed studies that the assumptions do not hold anymore and real time can be used? Then distributed applications could be implemented much easier because we do not need to care about message ordering anymore.

The Process Group approach has also inherent problems.

- Message delays are only considered until the messages are delivered to the application. But what about message delays by queuing in the application itself? This has an impact on the overall system performance because other messages my depend on an answer to the messages that have been sent earlier. I think this is an important practical problem. The designer of an system like ISIS is not necessarily aware of this problem because he is not involved in the application design. Therefore it is rather difficult to deal with this kind of delay. But after all the designer of a middleware has to work with some assumptions on the delay caused by the application. Otherwise it is necessary to develop the group communication layer application specific. But then the idea of a middleware that manages all the nasty things for the application programmer is gone. A possible solution would be to measure somehow the application specific delays and adjust the parameters of the middleware accordingly at runtime.

- If all messages are delivered to all nodes in the Process Group, then there is some overhead for the coordination of the single group members. Otherwise the same operation is done at all nodes which means that we do the same thing several times. But rather than ask how much an application has to pay for fault-tolerance, a more appropriate question is: At which level of replication outweighs the overhead the benefits of concurrency? We need to find the optimal level of replication at which we have an acceptable performance in case of failures and probably also gain or at least do not lose too much performance through concurrency.

If we consider real-time guarantees in mission-critical applications, Virtual Synchrony may help. It offers real-time guarantees in the following sense: Virtual Synchrony uses failure detectors. These failure detectors classify a distinct node as failed if it does not answer after some fixed amount of time. This means that a node receives a message during this amount of time or it never receives the message, because it is suspected to have failed.

References

[isis] K. Birman: *The Process Group Approach to Reliable Distributed Computing*; Communications of the ACM, 1993

[spinglass] K. Birman, R. van Renesse, W. Vogels: *Spinglass: Secure and Scalable Tools for Mission-Critical Computing*; DARPA DISCEX-2001