SHORT NOTES

Implementation of a Distributed Consensus Algorithm with OMNeT++

Andreas Dielacher, Thomas Handl, Christian Widtmann, Vienna University of Technology, Austria
{dielacher, handl, widtmann}@ecs.tuwien.ac.at

SNE Simulation Notes Europe SNE 19(1), 2009, 41-42, doi: 10.11128/sne.19.sn.09927

This paper examines the implementation of a simple distributed consensus algorithm with OMNET++. It will be shown that using this simulator *(i)* comes at nearly no cost and *(ii)* massively improves comprehension of the system under study.

Introduction

One of the major concerns of distributed computing is the ability of a group of nodes or processors to agree on a common value. This agreement is also called (distributed) consensus. Furthermore, in a realistic system model it is necessary to allow nodes to fail. The nature of failure depends on the fault model chosen. In the benign case, a node fails to send its messages in a consistent way (e.g. crash failures), whereas in the malicious case one has to deal with byzantine behavior. In the following we will focus on consensus with crash failures. This paper is structured as follows. Section 1 deals with the chosen simulator. We then describe the problem in Section 2 and our implementation in Section 3. The paper concludes in Section 4 with our results.

1 OMNeT++

OMNeT++ [2] is a modular open-source simulation environment with GUI support. It provides communication primitives allowing to easily model communication networks and alike although it has already been used in other areas like hardware architectures and business processes. This tool is used for scientific research as well as for industrial engineering. Examples for open source simulation models are network protocols like IP, IPv6 or MPLS.

From a technical point of view, OMNeT++ is a discrete and event driven simulator generator. The behavior of the system to be simulated is modeled using the well known C++ programming language. The structure of this system is described in a proprietary language called NED. OMNeT++ then compiles this code into a stand-alone simulator with GUI.

OMNeT++ offers many features such as user-defined message definition, message statistics, message tracking, visualization of network traffic and many more. Nevertheless, there are some inconveniences to handle. For example it is not possible to implement timeouts directly. This deficiency has to be overcome by using “self messages”, a usual approach in distributed computing models.

```
Initially $V = \{x\}$.
for round $k, 1 \le k \le f+1$ do
  send $v \in V : p_i$ has not already sent $v$ to all proc.
  receive $S_j$ from $p_j$, $0 \le j \le n-1$, $j \ne i$
    $V := V \cup \bigcup_{j=0}^{n-1} S_j$
    if $k = f+1$ then $y = \min(V)$
```

Algorithm 1. Simple consensus, code for one processor p_i

2 Distributed consensus with crash failures

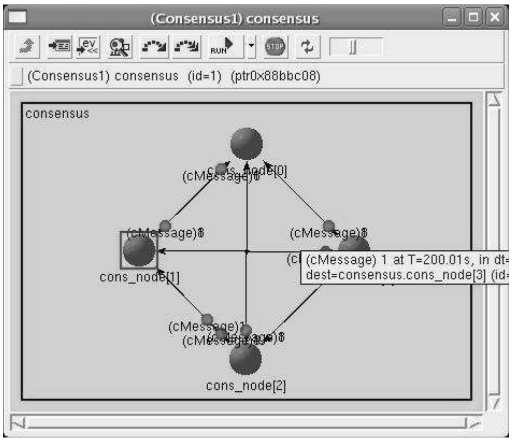
In this paper we focus on the standard synchronous model, where processors act in a round-based manner. Each round consists of sending and receiving one or more messages and one zero time computing step on each processor. Computation steps occur at the beginning of each round.

The algorithm we implemented is taken from [1] and listed in Algorithm 1. Each processor has a—probably distinct—value x. It is the intention of this algorithm to provide every processor with the same value after its completion. This indicates that nodes have to share their data and need common criteria to decide it. The algorithm operates such that in every round each processor sends a value it has not sent yet to all other processors. Each processor stores the received values in its set V. We allow one processor per round to fail with crash failure. In particular, we allow a failing node to send an arbitrary number of correct messages in the specific round it fails. After the crash a processor is not allowed to send any more messages. For this algorithm to work, the network has to be fully connected. It can be shown that the algorithm needs at least $f+1$ nodes, where f is the amount of crash failures tolerated.

3 Implementation

Knowledge of the C++ programming language can usually be presumed. So the main challenge in implementing a distributed algorithm in software usually turns out to be the comprehension that Algorithm 1 describes the behavior of one instance of a node whereas the system under study constitutes many nodes processing concurrently.

As shown in Figure 1 the GUI of the simulator created by OMNeT++ is quite intuitive. It can be seen that our model is made up of four nodes, thus tolerating three crash failures. Besides simulating the behavior of a fault-free net in the first place, we also deployed crashing nodes. In order to save compile time, we exploited a certain feature of OMNeT++: parameterization of the modeled system. By providing the number of nodes to crash as a parameter, arbitrary experiments can be conducted easily.



(cMessage) 1 at T=200.01s, in dt=
dest=consensus.cons_node[3] (id=

Figure 1. GUI of OMNeT++

The implementation of the behavior of this algorithm requires about 150 lines of code.

4 Conclusion

In this paper, we presented the successful implementation of a simple but nevertheless important building block of distributed computing: a consensus algorithm. In more detail, we have shown that the chosen simulator, OMNeT++, is *(i)* capable of simulating such an algorithm and *(ii)* particularly useful for the purpose of demonstration and also education. OMNeT++ has already proven to be very useful for the distributed computing community.

In a further step, we will extend the fault model to arbitrary (byzantine) faults and also examine data fusion algorithms. The results are intended to establish the basis for a hardware implementation of such an algorithm serving as a fault tolerance layer in distributed systems.

References

[47] H. Attiya, J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. Wiley Series on Parallel and Distributed Computing. Wiley and Sons, 2004.

[48] OMNeT++ Community Site. http://www.omnetpp.org

Corresponding author: Andreas Dielacher,
Department of Computer Engineering
Vienna University of Technology
Wiedner Hauptstraße 8-10, 1040 Vienna, Austria
dielacher@ecs.tuwien.ac.at

Received: October 12, 2008
Revised: January 20, 2009
Accepted: February 5, 2009