Report 4: Groupy

Peter Hamberg

September 29, 2011

1 Introduction

Making an atomic multicast group membership service.

We have a system were processes need to coordinate actions. At any time, any process could die, and the remaining processes should still be able to perform the same actions in the same order. Coordination and syncronization is the key here.

2 Main problems and solutions

The solution has two layers: the application layer and the multicast layer.

The application layer only knows of its multicast layer node. It simply sends and receives messages to and from the node. (As well as ok'ing any request for new nodes to join the group.)

The multicast layer is where the magic happens. When creating a new node and connecting it to the group, it queries a node already in the group for the group's current state and joins the fun. Since all nodes are ordered with regard to the time they joined the multicasting leader is always the oldest node. This node will receive messages from other nodes and multicast them to all the nodes in the correct order.

Since so much of the code for this assignment was given in advance and was responsible for the creation of several additional processes, it was very hard to automate testing for much of the code. I gave up that track rather quickly and decided to automate a manual testing function instead. The function will start a new node every eight ticks and destroy a random existing node every ten ticks, for some given arbitrary length of time 'tick'. This saves a lot of typing during testing, which is nice since typing into the erlang terminal is such a chore.

3 Evaluation

The three different solutions become more and more reliable and failiure-secure when going from gms1 through gms2 to gms3. While the first two versions are somewhat easy to desyncronize, the third will stand tall against simple node failiures. It still relies on the guaranteed delivery of messages between nodes and can be fooled into a bad state. Creating guards around this problem will of course require some increased complexity and overhead in the algorithm.

4 Conclusions

I appreciated this lab, since it forced an understanding of this weeks topic. The actual inner workings of the atomic multicast can be difficult to grasp at first, so it is good to be able to implement it in a practical setting. This gives a new angle of observation and helped me grasp some of the less understandable, but very useful aspects of the topic.

Actually, since more of the code for this week's lab was given in advance, it was harder - in a somewhat unsatisfactory way - to understand the problem and fix all the bugs.