**Zack Tillotson**

**CSCI586 Fault Tolerant Computing**

**Oct 25, 2010**

**Homework 2**

1. Assume a program takes T to finish, but has a failure rate of .

1. What is the expected program execution time, E?
2. If checkpoints are used to tolerate failures, what is the optimal checkpoint interval I?
3. If it takes one hour to checkpoint, what is the approximate I if MTTF is 1 day?
4. What is the optimal execution time E?
5. Is it possible to reduce E by 90% if tc is reduced by 1%?

**2. Explain the domino effect in poorly chosen checkpoints of a two process system.**

The effect of a failure in P at that point will be that the system must revert to the initial state and start over. This is due to a domino effect of alternating rollbacks in P and Q which create orphan messages each time.

When P fails it needs to rollback to checkpoint P3, however, this leaves message f an orphan, since Q has received it but P has not yet sent it at P3. In order to get to a consistent state, Q must rollback to before f was received, to checkpoint Q2. When Q goes to checkpoint Q2, however, message e is orphaned. In order to achieve a consistent state, P must then rollback to checkpoint P2. This in turn orphans message d, so Q rolls back to Q1. This continues with P going back to P1 because of c, Q going back to the initial state because of b, and then finally P going back to the initial state because of a.

**3. Given a 2 process system with the plan to take checkpoints every hour, and assuming the clocks are within 3 microseconds of each other and that messages take at least 1 microsecond to send, describe 2 approaches to prevent orphan messages in this system.**

Given process P, process Q, and message A being sent from P to Q, the only way message A can be orphaned is if P sends A to Q after P checkpoints but before Q checkpoints. This situation allows A to be orphaned because any failure after both checkpoints will be rolled back to the state where Q has received A, but P has not sent it. The first way to prevent this problem is from the sending processes point of view and the second is from the receiving processes point of view.

The first approach is based on limiting when processes can send messages during the period just before the specified checkpointing time. In order to prevent orphaning messages, a process which has just checkpointed and wants to send a message should wait until it’s sure the receiving process has checkpointed before sending the message. In order to guarantee this, a “blackout period” is imposed, during which no process is allowed to send a message. In our example, each process attempts to checkpoint at time τ. If the other processes’ clock is within time δ of the first processes’ clock, then by the time τ- δ we can be sure the other process has checkpointed. However, if we also assume our message takes at least time β to send, we may reduce the blackout period by this amount and still be ensured our message wont arrive until after the other process has been checkpointed. So, the final blackout period when a process is not allowed to send a message to other processes, is [τ, τ+δ-β]. In our example for τ = midnight, δ = 3 microseconds, and β = 1 microsecond, the blackout period = [midnight, midnight+2 microseconds].

The second approach is based on limiting when processes may **receive** messages. In this system, a process may send a message whenever it likes, but they aren’t accepted by the receiving process during the time right before the process checkpoints. This ensures that a received around the time of a checkpoint isn’t remembered (via the checkpoint) unless the sending process is sure to have checkpointed as well. If a process checkpoints at time τ, the first moment another process might have checkpointed is τ- δ, and the first moment we might get a message from that process is τ-δ+β, so our blackout period = [τ-δ+β, τ].