Exercise Session 6
NBAC, TRB, and VSC

November 12, 2012

Problem 1

Devise two algorithms that, without consensus, implement weaker specifications of NBAC *by replacing the termination property with the following ones:*

1. *Weak termination: Let p be a distinguished process, known to all other processes. If p does not crash then all correct processes eventually decide. Your algorithm may use a perfect failure detector.*
2. *Very weak termination: If no process crashes, then all processes decide. Is a failure detector needed to implement this algorithm?*

Solution

The first algorithm may rely on the globally known process p to enforce termination. The algorithm uses a perfect failure detector $\mathcal{P}$ and works as follows. All processes send their proposal over a point-to-point link to p. This process collects the proposals from all processes that $\mathcal{P}$ does not detect to have crashed. Once process p knows something from every process in the system, it may decide unilaterally. In particular, it decides COMMIT if all processes propose COMMIT and no process is detected by $\mathcal{P}$, and it decides ABORT otherwise, i.e., if some process proposes ABORT or is detected by $\mathcal{P}$ to have crashed. Process p then uses best-effort broadcast to send its decision to all processes. Any process that delivers the message with the decision from p decides accordingly. If p crashes, then all processes are blocked.

Of course, the algorithm could be improved in some cases, because the processes might figure out the decision by themselves, such as when p crashes after some correct process has decided, or when some correct process decides ABORT. However, the improvement does not always work: if all correct processes propose COMMIT but p crashes before any other process, then no correct process can decide. This algorithm is also known as the Two-Phase Commit (2PC) algorithm. It implements a variant of atomic commitment that is blocking.

The second algorithm is simpler because it only needs to satisfy termination if all processes are correct. All processes use best-effort broadcast to send their proposals to all processes. Every process waits to deliver proposals from all other processes. If a process obtains the proposal COMMIT from all processes, then it decides COMMIT; otherwise, it decides ABORT. Note that this algorithm does not make use of any failure detector.

Problem 2

Can we implement TRB with the eventually perfect failure detector $\diamond P$, if we assume that at least one process can crash?

Solution

The answer is no. Consider an instance *trb* of TRB with sender process *s*. We show that it is impossible to implement TRB from an eventually perfect failure-detector primitive $\diamond P$, if even one process can crash.

Consider an execution E_1 , in which process *s* crashes initially and observe the possible actions for some correct process *p*: due to the termination property of TRB, there must be a time T at which *p* trb-delivers $\perp$.

Consider a second execution E_2 that is similar to E_1 up to time T, except that the sender *s* is correct and *trb-broadcasts* some message *m*, but all communication messages to and from *s* are delayed until after time T. The failure detector behaves in E_2 as in E_1 until after time T. This is possible because the failure detector is only eventually perfect. Up to time T, process *p* cannot distinguish E_1 from E_2 and *trb-delivers* $\perp$. According to the *agreement* property of TRB, process *s* must *trb-deliver* as well, and *s* delivers exactly one message due to the *termination* property. But this contradicts the *validity* property of TRB, since *s* is correct, has *trb-broadcast* some message $m \neq \perp$, and must *trb-deliver m*.

Problem 3

In this problem we will change the *view-synchronous communication (VSC)* abstraction in order to allow joins of new processes. Answer to the following questions:

1. Are the properties of VSC (as given in the class) suitable to accommodate the joins of new processes. Why / Why not?
2. Change the properties of VSC, so that they allow for implementations that support the joins of new processes. (Hint: focus on the properties of *group membership*)

Solution

Solution 1.1

No, the properties are not suitable for joins. The most obvious property is Local Monotonicity. Joins imply that the set of correct processes in a view can increase, and this would break the local monotonicity property. Furthermore, Completeness and Accuracy only refer to crashes, without imposing any conditions on the correctness of joins.

Solution 1.2

First, we need to add a $\langle Join|p\rangle$ event to allow new processes to join the group. After a process emits such an event, we says that it requested to join. The VSC layer emits a $\langle JoinOk\rangle$ event to the application when it has successfully joined a view. The application can start emitting broadcast requests after it receives the *JoinOk* event.

Group membership properties

Let us first look at the four group membership properties.

View Monotonicity. The monotonicity property of VSC (GM1) ensures that the number of processes in a view decreases over time. Since new processes can join, this needs to change: We conside three possibilities:

- Get rid of this property entirely.
- Require that views do not change for nothing: If a process installs views (j, N) and $(j + 1, M)$, then $M \neq N$.

- Require that views do not oscillate (i.e., travel back in time): if a process p installs views (i, M) and (j, N) where $j > i$, $q \in M$, and $q \notin N$, then for all $k > j$, if p installs (j, O), then $q \notin O$.

With the second option, the new property ensures that consecutive views have different sets of processes, i.e., that the view cannot change if there is no change in the correct set of processes. Notice, however, that it is still possible for two views to have the same set of processes, e.g., if a processes joins and then crashes. It is also possible for a process to repeatedly be included and excluded from a view.

With the third option, once a process is excluded from a view it can never come back.

Uniform agreement. The uniform agreement property of VSC (GM2) ensures that all processes install the same sequence of view. We will keep this property.

Completeness. If we choose the third version of monotonicity, then we can keep the completeness property of the group membership abstraction. If we choose one of the first two, we need to make some changes: Because the sequence of views is no longer monotonic, we need to strengthen a bit the completeness property of VSC (GM3): If a process p crashes, then there is $i \in \mathbb{N}$ such that for all correct process q, if $j > i$ and q installs view (j, M), then $p \notin M$.

To ensure that processes which want to join eventually join a view, we add the following completeness property: If a correct process p requests to join, then there is an integer i such that every correct process eventually installs view (i, M) such that $p \in M$.

Accuracy. If a process p installs views (i, M) and $(i + 1, N)$ where $q \in M$ but $q \notin N$, then q has crashed.

On top of those properties, we will also require that a process is included in a view only if it requested so.

Validity. If some process installs a view (i, M) and some process q is in M, then q previously requested to join or $q \in \Pi$.

Broadcast properties

Let us now look at the broadcast properties of VSC. Those are the same of for reliable broadcast (RB1,2,3,4). We have two options: either a process which joins needs to "catch-up" on all previously delivered messages, or a new process can just start with the messages of the first view in which it is included.

If we choose the first option, then we can leave RB1,2,3,4 unchanged.

If we choose the second option, then we need to relax Agreement (RB4) so that a process need to deliver only the messages sent in the view to which it participates: If message m is delivered by some correct process in view (i, M), then m is eventually delivered by all the process belonging to M. This way, if $p \notin M$ then p does not have to deliver m.

View Synchrony

Finally, we will keep the View Synchrony (VS) property as is.