C Fault Tolerance and Message Complexity

We now discuss the fault tolerance and the message complexity of the different mechanisms presented in Chapter 4.

C.1 The Energy Concept

C.1.1 Fault Tolerance

In this protocol we have three participants:

- the application node, on which the application, i.e. the dependency object, resides, and from which the agent receives its new energy.
- the agent node, on which the agent resides.
- the communication channel, over which the request and the grant are sent.

We assume that no information is stored on stable storage, i.e. if the application node crashes, then the application is lost, and if the agent node crashes, then the agent is lost.

Since this protocol implements orphan detection, we examine the following: first, whether a failure of the underlying system can incorrectly remove an agent that is no orphan, and second, whether a failure of the underlying system can lead to the continued existence of an orphan agent that should be removed.

Let us first examine, whether the protocol can incorrectly remove an agent in the presence of failures. If the application node crashes, then the application is lost and the agent is an orphan. This will be detected after the agent has consumed its energy. If the agent node crashes, then the agent is lost, and no orphan exists. In both cases the protocol reacts correctly, i.e. is not influenced by the failure. When the communication channel crashes, i.e. if a network partition occurs, then the energy request cannot be delivered, and the agent is removed incorrectly. Thus the protocol depends on the availability of the communication channel to guarantee that no agent is incorrectly recognized as orphan.

Let us now investigate whether every orphan is recognized even in the presence of failures. An agent is an orphan when the dependency object, i.e. in this protocol the application, no longer exists. If the application node crashes before the energy request is received, then no grant is sent back, the agent is correctly recognized as an orphan, and removed. If the communication channel is broken, the same happens. Thus the protocol recognizes every orphan agent correctly.
C.1.2 Message Complexity

The message cost added by the protocol is 2 messages per agent per granted energy request, and 1 additional message for a denied request. If NAK messages are employed, 2 additional messages are needed. If we assume that the common usage of the energy concept is for agents that wait for something, e.g. for a specific change in a remote database, then the request can be piggy-backed on the message signalling the change. This reduces the message cost to 1 message if additional energy is granted and to 0 if denied.

C.2 The Path Concept

C.2.1 Fault Tolerance

In this protocol we have three participants:

- the anchor node, where the agent has been created originally and where the anchor resides.
- the agent node, on which the agent resides.
- the path connecting the anchor node and the agent node, consisting of a number of nodes on which the path proxies are stored and of communication channels between these nodes.

For our discussion we assume a path consisting of \( n \) path proxies on \( n \) nodes. This will allow us later to compare the availability of the different concepts. These \( n \) nodes, the anchor node and the agent node are connected via \( n+1 \) communication channels. Furthermore, as before, we assume that no information is stored on stable storage, i.e. if the application node crashes, then the application is lost, and if the agent node crashes, then the agent is lost.

Since this protocol implements functionality to locate an agent, we examine the following: first, the availability of the path information stored on the nodes, and second, the availability when we try to follow the path, i.e. when an agent is to be located.

The path information is stored in the respective nodes. Hence we have a series of \( n \) nodes for which we compute the availability. Since the proxy information is not stored on stable storage, it is lost after a crash, i.e. we have no repair functionality for the path. Thus the availability \( A(t) \) equals the reliability \( R(t) \). Let \( A_v(t) \) be the availability of a node \( v \). The availability \( A_p(t) \) of the path itself, i.e. the availability of the path proxies, can then be calculated by the following equation (see Appendix B.1, Equation B-3):

\[
A_p(t) = \prod_{v = 1}^{n} A_v(t) \quad \text{(Equation C-1)}
\]