From static to dynamic protocols: adapting timeouts for improved performance*

A. Casimiro[1], **M. Dixit**[1]

[1]University of Lisbon, Faculty of Sciences (FCUL)

{casim,mdixit}@di.fc.ul.pt

Abstract. *In asynchronous systems subject to process and network failures, distributed protocols often use more or less explicit timeouts to achieve progress. Since safety properties are guaranteed independently of the specific timeout value, timeout selection tends to be seen as an implementation detail. However, when network delays are unstable and susceptible to network contention, such as in wireless environments, it becomes important to dynamically adapt timeout values in order to address performance concerns. In this paper we discuss the problem of transforming static timeout-based protocols into dynamic ones, which can autonomically and dynamically select timeout values for improved performance. We propose a methodological approach and we present an example that illustrates how the methodology applies in practice.*

1. Introduction

In asynchronous distributed systems, one fundamental problem is to ensure progress of distributed protocols despite the possible occurrence of process crashes or network failures. A typical solution is to set a timeout, possibly implicitly within a failure detector, for preventing processes to wait indefinitely for messages that may never arrive. When the timeout expires, this leads the protocol to assume that a failure occurred and to decide accordingly. However, since the decision might be wrong (as the system could be just slow), special care must be taken to ensure that safety properties are nevertheless secured. In consequence, *the typical concern in the definition of protocols for asynchronous distributed systems is to ensure safety, while the problem of selecting a good timeout value is often ignored*. In this paper we argue that in some distributed systems it is important to address the problem of timeout selection for improving performance. First, we explain our reasoning and motivation. Then we propose an approach to transform static protocols into dynamic ones, in which timeouts are adaptive. Finally we provide a short example to illustrate how the approach may be applied to an existing protocol.

2. Motivation

Why is the problem of timeout selection often ignored? In general, this happens because of the following main reasons. First, sometimes timing assumptions are hidden from the protocol, encapsulated by higher level abstractions, making it impossible to directly deal with the issue of timeout selection. Second, independently of the specific timeout values, safety is preserved and only performance may become an issue. Finally, since in good

*This work was partially supported by the FCT, through project TRONE (CMU-PT/RNQ/0015/2009) and the Multiannual and CMU-Portugal programmes.

runs, when no timeout expires, performance is not affected by the specific timeout value, large timeouts tend to be selected, so that good runs will be the rule and faulty runs the exception. However, in this case, the trade-off is the price to pay for handling exceptions. Performance ends up being dependent on: a) how large timeouts must be set, and b) how frequent are still the exceptions.

Therefore, to reason about performance, a crucial issue is the characterization of the temporal behavior of the network on which the distributed protocol is executed. Local Area Networks (LANs) constitute a very favorable environment from a temporal perspective. Network delays are very small (in the order of microseconds), very stable across different LANs and independent of the communicating nodes, and they are not much affected by contention. In general, in LANs it is possible to assume small upper bounds for communication delays, which will hold with a very high probability. Therefore, distributed protocols for asynchronous systems can easily perform well in these environments, since it is not difficult to select appropriate timeout values. On the other hand, in large-scale networks (WANs) delays are much higher and they strongly depend on the specific end points. Moreover, delays are not so stable over time, in particular because routes may change dynamically and global load fluctuations also have some impact on observed delays. Therefore, a correct timeout selection, namely involving dynamic adaptation, becomes more important in these environments. The problem becomes even more relevant when considering the operation in wireless environments. Network delays are also much larger than in LANs, but in addition they are strongly exposed to the effects of contention. Varying the number of nodes that actively execute a distributed protocol and communicate within a single hop distance has a clear impact on observable delays. This is illustrated in Figure 1, which compares the round completion delay we observed when running a consensus protocol on a LAN in our lab, and on a wireless network in the Emulab testbed [White et al. 2002].

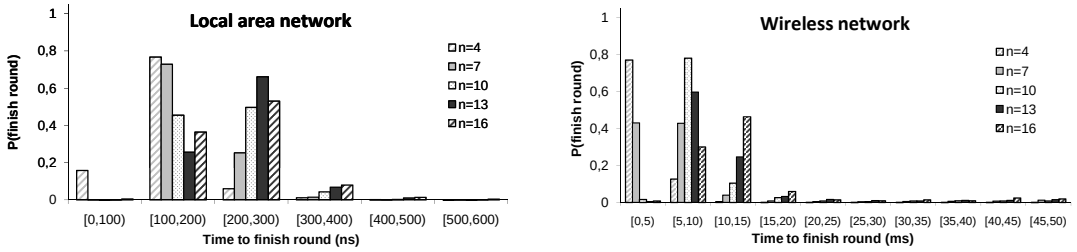

Figure 1. Protocol execution delays in a LAN and in a wireless network.

The previous discussion leads us to the main argument we defend and discuss in this paper. We argue that the problem of timeout selection, which has been treated as a minor issue in the context of distributed protocols for asynchronous systems, should be given particular attention when considering large-scale and wireless networks, for performance and practicality reasons. Furthermore, we believe that it may be possible to transform static protocols, which do not consider adaptive timeouts, into dynamic ones that perform better in environments of varying timeliness. This is what we discuss next.

3. A methodology for structured timeout adaptation

Given a static timeout-based protocol, the objective is to make it dynamic in the simplest possible way. We propose a modular approach, in which there is an independent service

that provides the right timeout to be used through a well-defined service interface. The *right* timeout depends on the specific protocol or application goals, since a smaller timeout leads to faster decisions, but increased probability of mistakes and a larger one helps avoiding mistakes, but prevents fast progress when faults indeed occur. In general, it may be necessary to translate performance objectives into requirements that the service can handle. Given that, we consider a composition involving three basic entities, described ahead: a timeout provisioning service, a mapping layer and the timeout-based protocol.

3.1. Timeout provisioning service

The timeout provisioning service must perform two main tasks. First, it must characterize the state of some relevant temporal variable (e.g. the network delay), based on observations of this variable. Based on that, the service can perform the second task, which is deriving a temporal bound (a timeout) that has a given probability to hold.

In our previous work we have designed *Adaptare* [Dixit et al. 2011], which is a probabilistic framework for dependable adaptation that may be used as a timeout provisioning service. *Adaptare* assumes that observed temporal variables can be described in probabilistic terms, that distributions change over time, but that they do not change arbitrarily fast. This allows to collect enough information for building useful probabilistic characterizations. The input to *Adaptare* are sample observations of the temporal variable (for instance, continuously measured network delays). Using these samples *Adaptare* applies several mechanisms to achieve a probabilistic characterization of the current state. Then, *Adaptare* can be used to obtain a bound (a timeout) that will hold with (at least) a specified probability, which is called coverage.

3.2. Mapping layer

In order to obtain a timeout from *Adaptare* it is necessary to provide the required coverage for the timeout. The coverage must be derived from performance requirements, although this might not be trivial. In the simplest case, it will be known that the best performance is achieved for a certain coverage. For instance, the performance of timeout-based failure detectors may be specified in terms of the mistake rate, which directly corresponds to the probability that a timeout expires too early. In this case, no specific mapping layer would be needed as the mistake rate would serve as the required coverage. In other cases, performance requirements are specified in terms of variables that cannot be directly translated into a coverage, like the execution time. In these cases, a mapping layer will be used to translate requirements into a coverage value, which will be bound to a specific protocol or application. In fact, the layer will implement a cost/benefit function expressed in terms of $\langle$coverage,timeout$\rangle$ pairs. These pairs are obtained from *Adaptare* whenever necessary, that is, when the protocol asks for a new timeout. When this happens, the mapping layer will run a procedure to determine if there is a new $\langle$coverage,timeout$\rangle$ pair that brings a relative improvement with respect to the currently considered $\langle$coverage,timeout$\rangle$ pair, when the two pairs are evaluated through the cost/benefit function. When the probabilistic characterization of the state provided by *Adaptare* changes, it is expected that a new timeout will be selected.

3.3. Timeout-based protocol

For transforming a static timeout-based protocol into a dynamic one, it is fundamental to first identify the timing variable of interest, on which the timeout is applied. For instance,

in a pull-based failure detector the variable of interest is the round-trip delay of the request/reply pair, while in some round-based protocol it may be the delay for completing a round. Then, the protocol must be instrumented for monitoring the temporal variable and for sending the measured values to *Adaptare*. Finally, whenever the timeout is used there must be a call to *Adaptare* for updating its value.

4. Application example

We applied the described approach for improving the performance of a consensus protocol designed for wireless environments [Moniz et al. 2009]. The protocol executes in rounds with several phases, in which every process sends a broadcast and waits for a given number of messages in order to make progress towards a consensus decision. A timeout prevents process to wait indefinitely, if no sufficient messages are received. In this case, processes start a new phase and resend the broadcast. In the original implementation, the timeout was fixed.

To be able to use an adaptive timeout, we first identified the temporal variable to be the time required for receiving a message, measured from the beginning of a new phase. The initial time of each phase is recorded and a delay is measured whenever a message pertaining to that phase is received, even if the process has already advanced to a new phase. All these delays are used to feed a local instance of *Adaptare*, which is thus able to characterize the time it takes to receive a message.

In this protocol, the performance objective is to reach fast consensus. This depends on the selected timeout as follows. If the timeout is made short, then either a phase completes fast (within the timeout), or the protocol retries a new phase shortly. The cost of making the timeout too small is that it becomes more difficult to complete phases timely and the network contention increases due to more frequent broadcasts. On the other hand, making the timeout large will delay decision when messages are late and the protocol keeps waiting for them (until the timeout expires). Given the protocol definition, it is possible to know how many messages it is necessary to receive in each phase in order to make progress. The remaining messages (if any) arriving after the timeout are irrelevant, since they will be discarded. Therefore, a good criteria for achieving optimized performance is to use a timeout that is adjusted to the number of messages one expects to receive. The coverage is thus set to the number of required messages over the total number of possibly received messages, which depends on the total number of processes.

We ran some experiments in a wireless environment (on Emulab) to compare implementations with static and dynamic timeouts. In particular, we were interested in knowing how the number of participating nodes would impact on performance. Therefore, the experiments were repeated for a varying number of nodes, ranging from 4 to 16. As expected, we were able to observe the positive effects of using an adaptive timeout. The most significant observations were the following. First, the average timeout value used in the dynamic implementation was increasing linearly from 13ms (with 4 processes) to about 25ms (with 16 processes). This is a direct consequence of the increased contention and communication delays observed when increasing the number of nodes. Second, we observed that the average latency of consensus was increasing much faster in the static implementation (twice faster). For 16 processes, the average latency of the static version was about 225ms, while it was just 114ms in the dynamic version.

Lastly, we also observed that the average number of broadcasts per round was kept almost stable in the dynamic version (between 3 and 6), while in the static version it increased from 4 to 23.

5. Conclusion

In this short paper we motivated for the need to consider adaptive versions of protocols in asynchronous and dynamic distributed systems, like in WANs or wireless networks. We proposed an approach that requires a minimum amount of changes to static protocols, while still allowing them to benefit from dynamic timeouts. A timeout provisioning service is key to the approach. When performance requirements cannot be directly expressed in probabilistic terms, a mapping layer is necessary to translate protocol requirements using a cost/benefit function. The approach was applied to transform a static consensus protocol for wireless environments and the results allowed us to conclude that it is possible to achieve the intended performance improvements.

References

Dixit, M., Casimiro, A., Lollini, P., Bondavalli, A., and Verissimo, P. (2011). Adaptare: Supporting automatic and dependable adaptation in dynamic environments. *ACM Transactions on Autonomous and Adaptive Systems*, (to appear). Also as Dep. of Informatics, Univ. of Lisboa, Technical report TR-09-19.

Moniz, H., Neves, N. F., Correia, M., and Verissimo, P. (2009). Randomization can be a healer: consensus with dynamic omission failures. In *Proceedings of the 23rd International Conference on Distributed Computing*, pages 63–77.

White, B., Lepreau, J., Stoller, L., Ricci, R., Guruprasad, S., Newbold, M., Hibler, M., Barb, C., and Joglekar, A. (2002). An integrated experimental environment for distributed systems and networks. In *5th Symposium on Operating Systems Design and Implementation*, pages 255–270, Boston, MA, USA.