A Survey on Failure Detection in Distributed Systems

John Sampson

Abstract—Originally developed to circumvent the FLP impossibility result, failure detection has proved to be a rich abstraction for understanding levels of synchrony between the asynchronous and synchronous models. This paper discusses some advantages of exploring this intermediate space in terms of failure detector augmentation instead of particular specifications of partial synchrony. Limitations of failure detection augmented asynchronous systems as compared to synchronous systems are then noted. The paper then examines some applications of failure detection to models of consensus featuring stronger failure modes than crashing as well as the application of failure detection to algorithms other than consensus. Some notable issues concerning implementing failure detectors in real systems are raised, with special attention given to performance. Finally, the continuing relevance of failure detection is briefly discussed.

I. ORIGINS OF THE FAILURE DETECTION APPROACH

The asynchronous model is very attractive in that it is sufficiently weak that it can be readily mapped as representative of a given real system. However, ever since the FLP impossibility result [1] showed that the simple asynchronous model is too weak to solve consensus, and thus most interesting algorithms, even under remarkably benign failure assumptions, interest in the asynchronous model has shifted to figuring out how to augment the asynchronous model so that it becomes capable of solving interesting problems. As the synchronous system model is known to be more powerful, but often too strict in its assumptions to be practical to implement, notions of partial synchrony were investigated. However, Chandra and Toueg [2], rather than relying on explicit parameters of partial synchrony, developed a model based on augmenting the asynchronous model with unreliable failure detectors, showing that this augmentation was sufficient to circumvent the FLP result. In particular, they introduced the notion of different classes of failure detectors and highlighted those that satisfied different pairs (see Table 1.) of the following completeness and accuracy axioms:

**Strong Completeness:** Eventually every process that crashes is permanently suspected by every correct process.

**Weak Completeness:** Eventually every process that crashes is permanently suspected by some correct process.

**Strong Accuracy:** No process is suspected before it crashes.

**Weak Accuracy:** Some correct process is never suspected.

**Eventual strong accuracy:** There is a time after which correct processes are never suspected.

**Eventual weak accuracy:** There is a time after which some correct process is never suspected.

In a companion paper [3], it was then shown that failure detectors of the $\diamond$W class were the minimal strength failure detectors needed to bypass FLP, given a majority of the processes being correct. This result is somewhat remarkable in that it shows that a failure detector that only satisfies incredibly weak assumptions can still change the expressive power of the model it augments in relevant ways.

<table>
<thead>
<tr>
<th>Completeness</th>
<th>Strong</th>
<th>Weak</th>
<th>Eventual Strong</th>
<th>Eventual Weak</th>
</tr>
</thead>
<tbody>
<tr>
<td>Strong</td>
<td>P</td>
<td>S</td>
<td>$\diamond$P</td>
<td>$\diamond$S</td>
</tr>
<tr>
<td>Weak</td>
<td>Q</td>
<td>W</td>
<td>$\diamond$Q</td>
<td>$\diamond$W</td>
</tr>
</tbody>
</table>

Table 1. A table of the failure detector classes introduced in Chandra and Toueg

II. ADVANTAGES OF THE FAILURE DETECTION APPROACH

While both partial synchrony models and the failure detection augmentation were enough to circumvent FLP, the failure detection approach had several advantages over relying on partial synchrony. Notably, it was less implementation constrained; the failure detector classes were defined via adherence to sets of axiomatic properties, and thus made no specific statements about how they needed to be implemented. Because of this form of specification, there exists a well-defined hierarchy of failure detectors. This hierarchy allows for a clean notion of reducibility, which is lacking in definitions of partial synchrony. Indeed, it is the ability to consider reductions to and from a failure detector of unspecified implementation that allows for the possibility of results such as the minimal nature of $\diamond$W, whereas the minimal explicit synchronicity necessary to avoid the FLP result is not known, despite there being many properties shown to be sufficient.

III. LIMITATIONS OF THE FAILURE DETECTION APPROACH

Unfortunately, it does not appear that the failure detection model suffices to entirely supplant notions of synchronicity. While failure detectors allow for an asynchronous system to combat the difficulty in distinguishing between slow and failed processes, even augmented with what Chandra and Toueg termed a “Perfect” failure detector, an asynchronous system remains a weaker model than a synchronous one, being unable to solve the Strongly Dependant Decision problem [4]. Moreover, even for problems that both models can solve, there appear to be shortcomings in the relative efficiency of the failure detector based solution. Thus, there is some real sense in which the failure detector model, in not dealing with bounded times, cannot capture the full power of a synchronous system; if nothing else, a synchronous system
can implement a time bounded perfect failure detector that is not only complete and accurate, but guaranteed to report failures no more than some fixed constant amount of time after they occur, which is a strength not attainable in the asynchronous model.

IV. APPLICATIONS BEYOND FLP

The FLP result was important because it addressed a particularly weak failure model, and thus applied to all stronger failure models. However, that a particular failure detection model allows the FLP result to be avoided does not directly imply anything about whether it is sufficient to overcome a stronger failure model. Rather, it is only by allowing for solutions in the strongest failure model that an augmentation becomes universally applicable. Thus, work has proceeded in determining the necessary augmentations for solving consensus in the asynchronous model in more complicated failure models, and in determining the minimal failure detection necessary to solve problems other than consensus.

A. More Complicated Failure Models

The crash failure model defies a certain intuition in that it requires all processes that crash to remain crashed for the duration of the algorithm run. It is more realistic to have to consider the possibility that some of these processes may restart and that message loss may occur. However, this complicates the failure model, as now processes that either intermittently experience omissions, or neither remain up nor remain crashed (i.e. are unstable) must be considered [5]. For the problem of consensus, even if the majority of processes never crash, the unstable processes represent an additional hurdle vs. crash failure, in that they may indefinitely postpone progress by happening to only be active at inopportune times. However, despite additional complexities, if stable storage is introduced, it is shown [5] that a class of failure detectors satisfying versions of the completeness and accuracy axioms that define $\diamond S$, those axioms having been modified slightly to incorporate the concept of an unstable process, are sufficient to solve consensus, even in situations where the majority of processes may, at some point, crash.

Notably more difficult is adapting failure detection to a Byzantine model. An immediate hurdle in supplying a failure detector for a Byzantine failure scenario is algorithmic dependence. While all crashed or omitting processes can be viewed as behaving in the same general manner, a process exhibiting Byzantine failure may only give evidence of having failed relative to implicit knowledge by other processes of how the failing process was expected to behave. Indeed, while crashed and omitting processes need failure detection to differentiate them from the slow correct processes, Byzantine failings must be additionally differentiated from the fast correct processes. Thus, when discussing failure detectors in a Byzantine environment, it is convenient to consider classes of failure detectors capable of being parameterized by the algorithm that requires their consultation [6]. Under this view, one may consider a process that is sending erroneous messages, with respect to the algorithm it was supposed to be implementing, to not be sending anything at all.

However, absent a parameterized view, which may, depending on the algorithm in question, not be practically attainable, one must split Byzantine faults into detectable and undetectable categories. Likewise, algorithmically valid messages may still be maliciously misleading. Fortunately, even if the failure detector can only address the detectable faults, the undetectable failures can be masked at the algorithmic level [7]. However, in introducing such a model, one must construct new axioms defining new positions in the failure detection hierarchy. Introducing the concept of a mute process [7] and the corresponding $\diamond M$ failure detector relies on the new “Mute Completeness” axiom supplanting the strong completeness axiom of Chandra and Toueg [2]. Likewise the $\diamond W(Byz)$ and $\diamond S(Byz)$ failure detector classes rely on “Eventual Strong Byzantine Completeness” substituting “detectable Byzantine fault” for “crash” and the more radically changed “Eventual Weak Byzantine (k+1)-Completeness” axiom, namely “there is a time after which every process that has exhibited a detectable Byzantine fault is permanently suspected by at least k+1 correct processes.” [8] It is notable that in both cases it is the completeness requirement that has changed, despite what might be seen as a potential for change in the accuracy constraint due to attempts of Byzantine processes to convince other processes that correct processes have failed [8]. It turns out that the changes are intimately linked, in that completeness had to change to allow for accuracy preserving transformations between weak and strong completeness. However, the truly relevant result is that, using these new axioms it was shown that there is a minimal failure detector class that allows consensus to be solved in authenticated systems featuring Byzantine failures in no more than a fixed portion of processes [8]. In particular, an approach is detailed for the case where N>3t.

In an asynchronous environment where the number of processes that may crash-fail is unbounded, particularly interesting results arise about the relative difficulties of problems, provided certain types of failure detectors are excluded, namely those that can guess the future. In such an environment, the hierarchy of detectors becomes somewhat collapsed [9] and S and P become only trivially different. Indeed, in this restricted environment, it can be shown that the collapsed P/S class of detector becomes the weakest detector necessary to solve both uniform consensus and terminating reliable broadcast, and non-uniform consensus, not requiring P, is shown as being distinct in difficulty from uniform consensus. Admittedly, removing either restriction eliminates the results, so the question arises as to how meaningful the restrictions are. Clearly, that there is no bound on the number of processes that fail is a valid, if pessimistic and limiting, assumption. As to the restriction on detectors, given that the term used by the authors to describe the set of allowed classes of detector is “Realistic” [9] it is clear what their viewpoint is; guessing the future is generally rather difficult, but the
particular phrasing also seems to exclude randomized guessing approaches, and one could argue that it is asking too much to declare randomness disjoint from realism.

B. Problems Other Than Consensus

Consensus, while important, is not the only problem for which asynchronous solutions are desired, nor the only one to which failure detection can be applied. Non-Blocking Atomic Commit (NBAC) is a problem that, like consensus, cannot be solved in the asynchronous model when there are crash failures. Moreover, if one restricts the set of failure detectors available to exclude those that are dependent on exact time, and the set of failed processes to not be in the majority, NBAC is shown to be strictly harder than consensus [10]. Key to the difference between consensus and NBAC is the ?P, or anonymously perfect, failure detector, which satisfies the following properties: "Anonymous Completeness: If some process crashes, then there is a time after which every correct process permanently detects a crash, and Anonymous Accuracy: No crash is detected unless some process crashes." [10]. More specifically, under the restricted model, ?P is exactly the difference between NBAC and consensus, with a combined ?P and ØS detector being the minimum strength failure detection needed to solve NBAC.

V. IMPLEMENTATION

While there is no doubt that the failure detection abstraction is a useful tool for formally describing the operating requirements of a system, without construction of failure detectors, it is not a practical tool. One of the first things facing an effort to construct a failure detector is the difficulty of untangling the failure detection from the application that will be using it [11]. It has been previously mentioned that, if the environment includes Byzantine failures, this is particularly difficult [6], and renders “Black-Box” semantics for the failure detector interface untenable. If one is able to do so, however, there are advantages to being able to make failure detectors distinct from the processes they monitor, such as being able to distribute information among the failure detectors in modes more bandwidth efficient and scalable than the underlying processes are themselves using to communicate [11], allowing for the convenience of cleanly having distinct failure models for the processes and for the failure detectors, and being able to limit, via interface, the degree to which even a Byzantine failure of a process can influence the set of processes suspected of having failed.

However, clean the interface, performance is still an overbearing factor in the usefulness of an implementation. Given that failure detector classes that make few assumptions about the underlying synchrony of the system tend to offer eventual guarantees only [12], it is important to consider the performance impact of relying on them in real systems with more stringent constraints. Certain performance pitfalls are possible when relying on failure detection. Incorrectly suspecting a process to have failed can be costly. In a rotating coordinator approach, for example, a poor (slow) choice of coordinator can lead to no progress occurring in any given round [13]. In group membership protocols, forcing a correct process out of a group is likewise very time expensive [14], as the rejoining operation has significant overhead. Thus, approaches that adaptively estimate the slowness of processes so as to make better choices of coordinators [13], approaches that introduce asymmetry between input and output triggered suspicions [14] to delay potentially expensive operations, and approaches championing the use of randomization in failure detection [12, 15], though not increasing the range of problems that can be solved, increase the degree to which problems can be solved quickly. As practical constraints [14] may cause distributed systems to have to perform some sort of group membership, and as group membership in turn relies on failure detection, failure detection performance is clearly an issue.

VI. DISCUSSION

Though any particular implementation of a failure detector tends to rely on some explicit assumption of partial synchrony, and it has been shown that even perfect failure detection does not itself encompass all of the power of synchronization, the failure detection methodology of augmenting the asynchronous model remains profoundly relevant in the degree of generality and completeness to which properties can be proved about failure detection augmented systems that is not available to specifications of partially synchronous systems. Because of the prevalence of systems designed around higher level protocols and algorithms that utilize failure detectors, what was introduced as a clean and powerful abstraction has given rise to a class of services whose support is a necessity in the implementation of many distributed systems. A sufficient number of properties have been proven about failure detectors now, that it does not seem undue that focus has shifted to their practicality, whether it be hinged on the only eventual nature of the failure detector’s promises of completeness or accuracy, on the efficiency of particular algorithmic approaches employed, or on the scalability of message requirements. Addressing these issues can only be assisted by disentangling, where possible, the problem being solved, whether it be consensus, group membership, or anything else, from the mechanisms being used to implement failure detection, and examining the properties exhibited by a failure detector that is itself a distinct distributed application.

REFERENCES

[4] Bernadette Charron-Bost, Rachid Guerraoui, and Andre Schiper. Synchronous system and perfect failure detector: Solvability and


