1. Open Questions on Consensus Performance in Well-Behaved Runs

I. Keidar[1] and S. Rajsbaum[2]

[1] Department of Electrical Engineering, The Technion, Haifa 32000, Israel
email: idish@ee.technion.ac.il
[2] Instituto de Matemáticas, UNAM, Mexico
email: rajsbaum@math.unam.mx

1.1 Consensus in the Partial Synchrony Model

We consider the *consensus* problem in a message-passing system where processes can crash: Each process has an input, and each correct process must decide on an output, such that all correct processes decide on the same output, and this output is the input of one of the processes. Consensus is an important building block for fault-tolerant systems.

It is well-known that consensus is not solvable in an asynchronous model even if only one process can crash [1.13]. However, real systems are not completely asynchronous. Some partially synchronous models [1.12, 1.10] where consensus is solvable better approximate real systems. We consider a *partial synchrony* model defined as follows [1.12] [1]: (1) processes have bounded drift clocks; (2) there are known bounds on processing times and message delays; and (3) less than half of the processes can crash. In addition, this model allows the system to be *unstable*, where the bounds in (2) do not hold for an unbounded but finite period, but it must eventually enter a *stable* period where the bounds do hold. A consensus algorithm for the partial synchrony model never violates safety, and guarantees liveness once the system becomes stable. Algorithms for this model are called indulgent in [1.16].

What can we say about the running time of consensus algorithms in a partial synchrony model? Unfortunately, even in the absence of failures, any consensus algorithm in this model is bound to have unbounded running times, by [1.13]. In this paper we propose a performance metric for algorithms in the partial synchrony model, and suggest some future research directions and open problems related to evaluating consensus algorithms using this metric.

In practice there are often extensive periods during which communication is timely and processes do not experience undue delays. That is, many runs are actually stable. In such runs, failures can be detected accurately using time-outs. We are interested in the running time of consensus algorithms for partially synchronous models under such benign circumstances. We focus on runs in which, from the very beginning, the network is stable. We will call such runs *well-behaved* if they are also failure-free. Since well-behaved runs

[1] This is very close to the model called *timed-asynchronous* in [1.10].

are common in practice, algorithm performance under such circumstances is significant. Note that an algorithm cannot know a priori that a run is going to be well-behaved, and thus cannot rely upon it.

We will evaluate algorithm running times in terms of the number of communication *steps* (some other authors call them *rounds*) an algorithm makes in the worst case before all processes decide in well-behaved, and sometimes more generally, stable runs.

1.2 Algorithms and Failure Detectors

There are several algorithms for partially synchronous models that decide in two steps in well-behaved runs, which as we shall see, is optimal. Most notably, (an optimized version of) Paxos [1.20], and others such as [1.24, 1.17].

Many of these consensus algorithms use oracle unreliable failure detectors [1.6] that abstract away the specific timing assumptions of the partial synchrony model, instead of directly using the specific timing bounds of the underlying system. The unreliable failure detectors, in turn, can be implemented in the partial synchrony model. Our interest in understanding the performance of consensus algorithms under common scenarios takes us to questions on the performance of failure detectors in the partial synchrony model. Notice that such unreliable failure detectors can provide arbitrary output for an arbitrary period of time (while the system is unstable), but eventually provide some useful semantics (when the system becomes stable).

Chandra and Toueg [1.6] define several classes of failure detectors whose output is a list of *suspected* processes. The natural way of detecting failures using timeouts in a partial synchrony model yields a failure detector called *eventually perfect*, or $\diamond\mathcal{P}$, that satisfies the following two properties. *Strong completeness:* there is a time after which every correct process permanently suspects every crashed process. *Eventually strong accuracy:* from some point on, every correct process is not suspected. It is remarkable that although $\diamond\mathcal{P}$ is indeed sufficient to solve consensus, it is not necessary. The so called $\diamond\mathcal{S}$ failure detector has been shown to be the weakest for solving consensus [1.5], and it is strictly weaker than $\diamond\mathcal{P}$. A $\diamond\mathcal{S}$ failure detector satisfies strong completeness and *Eventually weak accuracy:* there is a correct process p such that there is a time after which p is not suspected by any correct process.

Another example of a failure detector class is the *leader election* service $\diamond\Omega$ [1.5][2]. The output of a failure detector of class $\diamond\Omega$ is the identifier of one process, presumed to be the leader. Initially, a failure detector of this class can name a faulty process as the leader, and can name different leaders at different processes. However, eventually it must give all the processes the same output, which must be the identifier of a correct process. In [1.5], it

[2] Originally called Ω, but we add the $\diamond$ prefix for consistency with the notation of the other failure detectors.

is shown that $\diamond\Omega$ is equivalent to $\diamond\mathcal{S}$, and hence weaker than $\diamond\mathcal{P}$. In a *well-behaved* run, a $\diamond\Omega$ failure detector announces the same correct leader at all the processes from the beginning of the run. We pose the following **open problems**:

- What is the weakest timing model where $\diamond\mathcal{S}$ and/or $\diamond\Omega$ are implementable but $\diamond\mathcal{P}$ is not?
- Is building $\diamond\mathcal{P}$ more "costly" than $\diamond\mathcal{S}$ and/or $\diamond\Omega$? Under what cost metric?

1.3 Lower Bound

Distributed systems folklore suggests that every fault tolerant algorithm in a partial synchrony model must take at least two steps before all the processes decide, even in well-behaved runs. We formalized and proved this folklore theorem in [1.19]. Specifically, we showed that any consensus algorithm for a partial synchrony model where at least two processes can crash will have at least one well-behaved run in which it takes two steps for all the processes to decide.

This is in contrast to what happens in a *synchronous* crash failure model: in this model there are consensus algorithms that, in failure-free runs, decide within one step. More generally, early deciding algorithms have all the processes decide within $f+1$ steps in every run involving f crash failures. This is optimal: every consensus algorithm for the synchronous crash failure model will have runs with f failures that take $f+1$ steps before all the processes decide [1.21].

Why then do consensus algorithms for the partial synchrony model require two steps in well-behaved runs, that look exactly like runs of a synchronous system? The need for an additional step stems from the fact that, in the partial synchrony model, a correct process can be mistaken for a faulty one. This requires consensus algorithms to avoid disagreement with processes that seem faulty. In contrast, consensus in the synchronous model requires only that correct processes agree upon the same value, and allows for disagreement with faulty ones. The *uniform consensus* problem strengthens consensus to require that every two processes (correct or faulty) that decide must decide on the same value. Interestingly, uniform consensus requires two steps in the absence of failures in the synchronous model, as long as two or more processes can crash, as proven in [1.7, 1.19]. The two step lower bound for consensus in the partial synchrony model stems from the fact that in this model, any algorithm that solves consensus, also solves uniform consensus [1.15].

The observation above – that an additional step is needed in order to avoid disagreement with processes that are incorrectly suspected to have failed – suggests the use of an *optimistic* approach to relax the requirement that two correct processes never decide on different values. An algorithm solving variant of consensus that is allowed to violate agreement in cases of false suspicions, can always terminate in a single step. Such a service can be useful for optimistic replication, if false suspicions are rare, and if the inconsistencies introduced in cases of false suspicions can later be detected and reconciled (or rolled-back). Group communication systems [1.9] such as Isis [1.4] take such an optimistic approach: they implement totally ordered multicast in a single step. If a correct process is suspected, inconsistencies can occur. Isis resolves such inconsistencies by forcing the suspected process to fail and re-incarnate itself as a new process, whereby it adopts the state of the other replicas. Another example is optimistic *atomic commitment* [1.18], which can lead to *rollback* in case of false suspicions. Our observation suggests that the likelihood of inconsistencies depends on the frequency of false suspicions, and leads to the following **open problem**:

- Formalize the notion of *likelihood of inconsistencies* for an application of optimistic consensus (or of a group communication system), and quantify its cost. Then, use this measure to analyze the cost-effectiveness of employing optimism in a particular setting.

1.4 Extensions and Future Directions

As a first extension, it is interesting to look at weaker notions of well-behavedness. E.g., consider runs where the system is stable but there are f failures. All the algorithms mentioned above can take as many as $2f + 2$ steps in such runs. Dutta and Guerraoui [1.11] present an algorithm that decides in $t + 2$ steps in stable runs, where t is the maximum number of possible failures, and show a corresponding lower bound. It remains an **open problem** to devise an algorithm that takes $f + 2$ steps for every $f <= t$. Such an algorithm would be optimal.

Another **future direction** is to consider runs that are initially unstable and then become stable, and to find lower and upper bounds on the time it takes to reach decision once stability is reached. A similar **open question** can be posed for asynchronous *self-stabilizing* algorithms.

So far, we have focused on algorithms and lower bounds stated in message-passing models. Similar algorithms exist in shared memory models, e.g., a version of Paxos for a shared memory model with read-write registers [1.14]; and another for a model with infinitely many processes and shared read-modify-write registers [1.8]. It is interesting to consider the meaning of our performance metric, namely, number of steps in well-behaved runs, in shared memory models. A common notion of "well-behavedness" in shared memory models is the absence of contention [1.1]. At a first glance, this may seem unrelated to our notion of well-behaved runs being synchronous failure-free runs. However, a deeper look reveals that the two notions are indeed related: in shared-memory versions of Paxos, absence of contention occurs when there is a single leader trying to propose a consensus value. This is akin to having a single leader in message-passing versions of Paxos. That is, the failure detector $\diamond\Omega$ enforces the absence of contention. Given this observation, we pose the following **open problems**:

– Are there formal generic transformations from well-behaved runs as defined herein to contention-free runs in different shared memory models?
– Are there generic complexity-preserving reductions from message passing to shared memory models? Do they preserve the communication step metric and some notion of "well-behaved runs"?

We have analyzed performance in terms of the number of steps an algorithm takes. Lower and upper bounds on the actual running time of consensus were postulated in a variant of the partial synchrony model that is stable from the outset [1.2]. An **open question** is to revisit these bounds, perhaps using layered analysis in the sense of [1.22], and to extend them to well-behaved runs.

Finally, [1.3] points out limitations of the communication steps performance measure in environments like the Internet, where message delays exhibit high variability. The running time of a communication step can depend heavily on the number of messages sent in the step, and the particular links over which messages are sent. This is due to the high variability of message delays (see, e.g., the analysis in [1.23]). A **future direction** is finding performance metrics that better capture algorithm performance in practice.

References

1.1 R. Alur and G. Taubenfeld. Contention-free complexity of shared memory algorithms. *Inform. Comput.*, 126(1):62–73, 1996.
1.2 H. Attiya, C. Dwork, N. Lynch, and L. Stockmeyer. Bounds on the time to reach agreement in the presence of timing uncertainty. *J. ACM*, 41(1):122–152, 1994.
1.3 O. Bakr and I. Keidar. Evaluating the running time of a communication round over the Internet. In *ACM Symp. on Prin. of Dist. Comp. (PODC)*, pp. 243–252, July 2002.
1.4 K. Birman, A. Schiper, and P. Stephenson. Lightweight causal and atomic group multicast. *ACM Trans. Comp. Sys.*, 9(3):272–314, 1991.
1.5 T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722, 1996.
1.6 T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, March 1996.
1.7 B. Charron-Bost and A. Schiper. Uniform consensus is harder than consensus (extended abstract). Tech Rep DSC/2000/028, EPFL, Switzerland, May 2000. Submitted to *J. Algorithms.*
1.8 G. Chockler and D. Malkhi. Active Disk Paxos with infinitely many processes. In *21st ACM Symp. on Prin. of Dist. Comp. (PODC)*, Jul 2002.
1.9 G. V. Chockler, I. Keidar, and R. Vitenberg. Group Communication Specifications: A Comprehensive Study. *ACM Comp. Surveys*, 33(4):1–43, December 2001.

1.10 F. Cristian and C. Fetzer. The timed asynchronous distributed system model. *IEEE Trans. Par. Dist. Sys.*, pages 642–657, June 1999.
1.11 P. Dutta and E. Guerraoui. The inherent price of indulgence. In *21st ACM Symp. on Prin. of Dist. Comp. (PODC)*, July 2002.
1.12 C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, Apr 1988.
1.13 M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32:374–382, Apr 1985.
1.14 E. Gafni and L. Lamport. Disk paxos. In *Int'l Symp. on DISt. Comp. (DISC)*, pp. 330–344, 2000.
1.15 R. Guerraoui. Revisiting the relationship between non-blocking atomic commitment and consensus. In *Int'l Wshop on Dist. Alg. (WDAG)*, pp. 87–100. Sep 1995. LNCS 972.
1.16 R. Guerraoui. Indulgent Algorithms. In *19th ACM Symp. on Prin. of Dist. Comp. (PODC)*, pp. 289–297. 2000.
1.17 M. Hurfin and M. Raynal. A simple and fast asynchronous consensus protocol based on a weak failure detector. *Dist. Comp.*, 12(4), 1999.
1.18 R. Jimenez-Peris, M. Patino-Martinez, G. Alonso, and S. Arevalo. A low latency non-blocking commit protocol. In *Int'l Symp. on DISt. Comp. (DISC)*, Oct 2001.
1.19 I. Keidar and S. Rajsbaum. A simple proof of the uniform consensus synchronous lower bound. *Information Processing Letters*, 2002. To appear.
1.20 L. Lamport. The part-time parliament. *ACM Trans. Comp. Sys.*, 16(2):133–169, May 1998.
1.21 L. Lamport and M. Fischer. Byzantine generals and transaction commit protocols. Tech Rep 62, SRI Int'l, Apr 1982.
1.22 Y. Moses and S. Rajsbaum. A layered analysis of consensus. *SIAM J. Comp.*, 31(4):989–1021, 2002.
1.23 S. Rajsbaum and M. Sidi. On the performance of synchronized programs in distributed networks with random processing times and transmission delays. *IEEE Trans. Par. Dist. Sys.*, 5(9):939–950, 1994.
1.24 A. Schiper. Early consensus in an asynchronous system with a weak failure detector. *Dist. Comp.*, 10(3):149–157, 1997.
1.25 D. Skeen. Nonblocking commit protocols. In *ACM SIGMOD Int'l Symp. on Management of Data*, pp. 133–142, 1981.