Timeliness, Failure-Detectors, and Consensus Performance

ALEXANDER SHRAER

Timeliness, Failure-Detectors, and Consensus Performance

Research Thesis

Submitted in Partial Fulfillment of the Requirements
for the Degree of Master of Science in Computer Science

ALEXANDER SHRAER

Submitted to the Senate of the Technion – Israel Institute of Technology
NISAN 5766 HAIFA April 2006

The Research Was Done Under the Supervision of Dr. Idit Keidar from the Department of Electrical Engineering, Technion, in the Department of Computer Science, Technion

THE GENEROUS FINANCIAL HELP OF THE TECHNION — ISRAEL INSTITUTE OF TECHNOLOGY IS GRATEFULLY ACKNOWLEDGED

Acknowledgments

I would like to express gratitude to the following people who helped me so much in completing the master degree.
My advisor Dr. Idit Keidar, for introducing the problem to me, sharing her vast knowledge and experience, and for her valuable guidance and support. Thanks for always willing to make time for me, despite more than a handful of other obligations.
I thank Marcos Aguilera, Partha Dutta, Rachid Guerraoui, Eshcar Hilel, Liran Katzir, Denis Krivitski, Keith Marzullo, and Neeraj Suri for many helpful discussions about the paper that led to this thesis.
Many thanks go to my thesis examinators - Prof. Hagit Attiya, Prof. Faith E. Fich and Prof. Yoram Moses, for their valuable input to my work.
Thanks to my mom, Zoya, without whom I wouldn't be where I am today.
And last, but certainly not least, to my wonderful girlfriend, Tanya, for all her support and encouragement, and for willing to put up with the long hours that this work required.

Contents

List of Figures

List of Tables

Abstract

Consensus is a widely-studied fundamental problem in distributed computing, theory and practice. Roughly speaking, it allows processes to agree on a common output. We are interested in the performance of consensus algorithms in different timing models. It is known that consensus is not solvable in a completely asynchronous environment, and a completely synchronous one is often too restrictive for real systems. Therefore, many middle-ground models have been previously suggested. One known model is the Eventually Synchronous model, in which the system is asynchronous for an arbitrary period of time and then becomes synchronous. Another method is assuming that each process has an unreliable failure detector oracle, which eventually gives some level of indication to the process about the failed processes or about a common non-faulty leader.

We study the implications that various timeliness and failure detector assumptions have on the performance of consensus algorithms that exploit them. We present a general framework, GIRAF, for expressing such assumptions, and reasoning about the performance of indulgent algorithms (algorithms that tolerate an arbitrary period of asynchrony). This framework addresses several shortcommings of a previously known framework - RRFD. We use GIRAF to revisit the notion of oracle (or model) reducibility and define α-reducibility that takes time complexity of the reduction into account. We investigate several interesting indulgent models (all weaker than Eventual Synchrony) using GIRAF and give upper and lower bounds for the number of rounds needed to reach consensus in these models.

Our results have several implications to the understanding of indulgent systems. First, we prove that Eventual Synchrony is not needed to achieve optimal consensus performance. We prove that the $\Diamond$S and the Ω failure detectors that are known to be equivalent in the literature, are very different when it comes to performance of consensus algorithms. We show an algorithm that works in an oracle-free model that is weaker than Eventual Synchrony but achieves very close performance to what is known to be possible in Eventual Synchrony. Finally, we show that several recently suggested models, and even much stronger models, are too weak to allow bounded-time consensus decision.

Chapter 1

Introduction

1.1 Background and Motivation

Consensus is a widely-studied fundamental problem in distributed computing, theory and practice. Roughly speaking, it allows processes to agree on a common output. We are interested in the performance of consensus algorithms in different timing models.

Although the synchronous model provides a convenient programming framework, it is often too restrictive, as it requires implementations to use very conservative timeouts to ensure that messages are *never* late. For example, in some practical settings, there is a difference of two orders of magnitude between average and maximum message latencies [9, 7]. Therefore, a system design that does not rely on strict synchrony is often advocated [37, 26, 12]; algorithms that tolerate arbitrary periods of asynchrony are called *indulgent* [30].

As it is well-known that consensus is not solvable in asynchronous systems [27], the feasibility of indulgent consensus is contingent on additional assumptions. More specifically, such a system may be asynchronous for an unbounded period of time, but eventually reaches *Global Stabilization Time (GST)* [26], following which certain properties hold. These properties can be expressed in terms of eventual timeliness of communication links [26, 18][1], or using the abstraction of *oracle failure detectors* [12]. Protocols in such models usually progress in asynchronous *rounds*, where, in each round, a process

[1] A timely link delivers messages with a bounded latency; the bound is either known or unknown a priori.

sends messages (often to all processes), then receives messages while waiting for some condition expressed as a timeout or as the oracle's output, and finally performs local processing.

Recent work has focused on weakening post-GST synchrony assumptions [32, 1, 2, 3, 43], e.g., by only requiring one process to have timely communication with other processes after GST. Clearly, weakening timeliness requirements is desirable, as this makes it easier to meet them. For example, given a good choice of a leader process, it is possible to choose a fairly small timeout, so that the leader is almost always able to communicate with all processes before the timeout expires, whereas having each process usually succeed to communicate with *every* other process requires a much larger timeout [7, 6]. In general, the weaker the eventual timeliness properties assumed by an algorithm are, the shorter the timeouts its implementation needs to use, and the faster its communication rounds can be.

Unfortunately, faster communication rounds do not necessarily imply faster consensus decision; the latter also depends on the number of rounds a protocol employs. A stronger model, although more costly to implement, may allow faster decision after GST. Moreover, although formally modeled as holding from GST to eternity, in practice, properties need only hold "long enough" for the algorithm to solve the problem (e.g., consensus) [26]. But how much time is "enough" depends on how quickly consensus can be solved based on these assumptions. Satisfying a weak property for a long time may be more difficult than satisfying a stronger property for a short time. Therefore, before choosing timeliness or failure detector assumptions to base a system upon, one must understand the implications these assumptions have on the running time of consensus. This is precisely the challenge we seek to address in this thesis.

1.2 GIRAF – A General Round-Based Algorithm Framework

This question got little attention in the literature, perhaps due to the lack of a uniform framework for comparing the performance of asynchronous algorithms that use very different assumptions. Thus, the first contribution of our work is in introducing a general framework for answering such questions. In Chapter 3.2, we present GIRAF, a new abstraction of round-based algorithms, which separates an algorithm's computation (in each round) from its round waiting condition. The former is controlled by the algorithm, whereas the latter is determined by an environment that satisfies the specified timeliness or failure detector properties. We give a time-free definition of timely links. Informally, a link is timely in a round, if messages sent in this round to a correct destination arrive in the same round. In addition, the environment can provide additional "oracle output" information for the protocol. In general, rounds are not synchronized among processes. GIRAF is inspired by Gafni's round-by-round failure detector (RRFD) [28], but extends it to allow for more expressiveness in specifying round properties, in that the oracle output can have an arbitrary range and not just a suspect list as in [28], and our rounds do not have to be synchronized among processes or be communication-closed like Gafni's[2].

One model that we study and cannot be expressed in RRFD ensures that each process receives messages from a majority, and in addition, provides an eventual *leader oracle*, Ω, which eventually outputs the same correct *leader* process at all processes; many consensus algorithms were designed for this model, e.g., [37, 22, 31]. Note that in order to ensure communication with a majority in each round, RRFD's suspect list must include at most a minority of the processes, and hence cannot, by itself, indicate which of the unsuspected processes is the leader. Thus, additional oracle output is required. In general, the question of which systems can be implemented in RRFD was left open [28]. In contrast, we show that GIRAF is *general* enough to faithfully capture *any* oracle-based asynchronous algorithm in the model of [11] (see Chapter 5). Note that GIRAF can be used to express models assuming various failure patterns and is not constrained to the crash failure model (even though the models we define in this thesis do assume crash failures). Moreover, GIRAF can be used to study problems other than consensus.

Since we focus on round-based computations, we replace the notion of GST with the notion of a *Global Stabilization Round (GSR)* [21]. Each run eventually reaches a round GSR, after which no process fails, and all "eventual" properties hold. More specifically, an environment in our model is defined using two types of properties: (1) *perpetual*

[2]In communication-closed rounds, each message arrives in the round in which it is sent.

properties, which hold in all rounds; and (2) *eventual properties*, which hold from GSR onward. The eventual counterpart of a perpetual property ϕ is denoted $\Diamond\phi$.

Since one can define different round properties, and organize algorithms into rounds where these properties hold, one can prove upper and lower bounds for the number of rounds of different types (i.e., satisfying different properties such as all-to-all communication in each round or communication with a majority in each round) that are sufficient and necessary for consensus. Note that, in order to deduce which algorithm is best for a given network setting, this analysis should be complemented with a measurement study of the cost of rounds of different types in that specific setting. The latter is highly dependent on the particularities of the given network and has no general answers (e.g., in a LAN, all-to-all communication may well cost the same as communication with majority, whereas in a WAN it clearly does not [7, 6]). GIRAF provides a generic analysis, which can be combined with a network-specific measurements to get a network-specific bottom line.

While GIRAF can express every asynchronous message passing algorithm, it does not necessarily provide meaningful information about all algorithms. Some algorithms are not naturally round-based and expressing them as such might not benefit their analysis. But in many cases, we believe that GIRAF can significantly improve the formulation and analysis of algorithms. Separating the computation from round waiting condition has several advantages. First, the computation code becomes clear as waiting conditions are eliminated from it. Second, the definition of a round becomes clear, as one needs now to give a separate specification for the round termination (i.e., waiting) condition. In many algorithms in the literature, the definition of a round is ad hoc, and is only implied by the computation code, leading to unclarity of what a round actually is. For example, Dobre and Suri [19] show that in some settings, an eventually perfect failure detector can lead to decision within two rounds, whereas algorithms using an Ω failure detector require at least three. By closely reading the proofs, we observe that the authors intend rounds in stable runs of the first model to include arrivals of messages from all processes, whereas the Ω-based model ensures only the arrival of messages from a majority of processes (including the leader) in each round, including in stable runs. However, this is only evident from reading the proofs. Moreover, such an analysis conceals the fact that different types of rounds are being counted in each case. Using GIRAF, this result can be explicitly stated as a tradeoff between two rounds of one well-defined type versus three rounds of another.

We use GIRAF to revisit the notion of oracle (or model) reducibility. Traditionally, reducibility between distributed computation models defines when one model can be implemented in another [12, 11], without taking complexity into account. In Section 4, we define the notion of α-*reducibility*, where round $GSR + l$ of the emulated model (for any l) occurs at most in round $GSR + \alpha(l)$ in the original model, in the same run. This notion captures reductions that incur a function $\alpha()$ penalty in running time. We define a special case of this notion, namely k-*round reducibility*, which is simply α-reducibility with $\alpha(l) = l + k$, and captures reductions that incur a k-round penalty in running time. Gafni [28] has posed as an open problem the question of finding a notion of an equivalence relation between models with regard to the extent in which one model "resembles" another. We hope that our notion of α-reducibility (and k-round reducibility) provides a convenient instrument to describe such relations.

1.3 Results

We use GIRAF to analyze consensus performance in different models. In this thesis, we consider a crash-failure model, where up to $t < n/2$ out of n processes may crash (before GSR). Our performance measure is the number of rounds until *global decision*, i.e., until all correct processes decide, after GSR.

Dutta et al. [21] have shown that in the *Eventual Synchrony (ES)* [26] model, where all links are timely from GSR onward, GSR+2, i.e., three rounds including round GSR, is a tight lower bound for global decision. We are interested in the implications of weakening the ES assumptions. Following the observation that in some settings communication with a leader or a majority can be achieved with significantly shorter timeouts than required for timely communication with all processes [7, 6], we focus on leader-based and majority-based models.

The first model we define is *Eventual Leader-Majority*, $\Diamond$LM (see Table 1.1, row 2). In this model, processes are equipped with a *leader oracle*, Ω [11]. We further require that the leader be a $\Diamond n$-source, where a process p is a $\Diamond j$-source if it has j timely outgoing links in every round starting from GSR [2] (the j recipients include p, and are not required to be correct)[3]. Finally, we require that each correct process eventually have timely incoming links from a majority of correct processes (including itself) in each round; this property is denoted $\Diamond(\lfloor \frac{n}{2} \rfloor + 1)$-destination$_v$, where the subscript v denotes that the incoming links of a process can change in each round. $\Diamond LM$ does not impose any restrictions on the environment before GSR. One might expect that weakening the ES model in this way would hamper the running time. Surprisingly, in Chapter 6, we present a leader-based consensus algorithm for $\Diamond$LM, which achieves the tight bound for ES, i.e., global decision by GSR+2. Our result suggests that eventually perfect failure detection is not required for optimal performance; Ω and timely communication with a majority suffice.

We then turn to see whether we can replace Ω with an equivalent (in the "classical" sense) failure detector, $\Diamond S$ [12]. $\Diamond S$ outputs a list of *suspected* processes at each process, so that eventually, every correct process suspects every faulty process, and there exists one correct process that is not suspected by any process. In Chapter 7, we show that in runs in which less than $\frac{n}{2} - 1$ processes fail, replacing Ω with $\Diamond S$ entails a lower bound of n rounds from GSR onward. In the literature, $\Diamond S$ is typically used with the assumptions that links are reliable (although not timely), and that messages from a majority arrive in each round, including before GSR [12, 44]. We therefore prove our lower bound for a stronger model, which we call *Eventually Strong-Reliable*, $\Diamond$SR. In this model, all links are reliable, processes are equipped with a $\Diamond S$ failure detector, the unsuspected correct process is a $\Diamond n$-source, and every process is a perpetual $(n - f - 1)$-destination$_v$, where $f \leq t$ is the number of actual failures in a given run, which is at least a majority whenever $f < \frac{n}{2} - 1$ (see Table 1.1, row 3).

The Ω oracle is clearly a powerful tool; our result for $\Diamond LM$ shows that Ω effectively eliminates the need for communication with all processes, and renders communication with majority sufficient. This raises the question of whether timely communication with a majority can suffice for constant-time consensus even *without* an oracle. To answer this question, we define the *Eventual All-from-Majority*, $\Diamond$AFM, model, where each correct process eventually has incoming timely links from a majority of processes, and outgoing

[3]In [2], the link from p to itself is not counted; hence a j-source in our terminology is a $(j - 1)$-source in theirs.

Model	Model Properties	Upper Bound	Lower Bound
ES	$\forall k \geq GSR$, all links are timely in round k	GSR+2 [21]	GSR+2 [21]
$\Diamond$LM	Ω, the leader is $\Diamond n$-source every correct process is $\Diamond(\lfloor \frac{n}{2} \rfloor + 1)$-destination$_v$	GSR+2 **Chapter 6**	GSR+2 [21]
$\Diamond$SR	$\Diamond S$, the unsuspected process is $\Diamond n$-source, every process is $(n-f-1)$-destination$_v$ $f < n/2 - 1$ and reliable links	GSR+$2n+2$ [44]	GSR+$n-1$ **Chapter 7**
$\Diamond$AFM	$\exists m \in N, f \leq m < n/2$ s.t. every correct process is $\Diamond(m+1)$-source$_v$ and $\Diamond(n-m)$-destination$_v$	if $n = 2m+1$ and $GSR > 0$: GSR+4 Otherwise: GSR+5 **Chapter 8**	GSR+2 [21]
$\Diamond$MFM(m) $m \in N^+$ $f \leq m < n/2$	every correct process is $\Diamond(n-m)$-source m correct processes are $\Diamond n$-source $(n-m)$ correct processes are $\Diamond(n-m)$-accessible every correct process is $\Diamond m$-accessible reliable links	Unbounded [2, 3, 43]	Unbounded **Chapter 9**

Table 1.1: Upper and lower bounds on consensus global decision times in various models ($t < n/2$).

timely links to a (possibly different) majority, including itself. It is possible for processes to have fewer outgoing links, and instead additional incoming ones (see Table 1.1, row 4). In Chapter 8, we give a consensus algorithm for this model that decides in constant time after GSR (five or six rounds, depending on the number of outgoing versus incoming timely links).We are not aware of any previous algorithm for the $\Diamond AFM$ model, nor any other constant-time (from GSR) oracle-free algorithm in a timing model other than ES.

Next, we examine whether one can weaken the model even further. Can we relax the assumption that *all* correct processes have timely incoming links from a majority, and allow a minority of the processes to each have one fewer timely link? (In case $n = 3$, only one timely link is removed). In Chapter 9, we show that the answer to this question is *no*, as this renders the problem unsolvable in bounded time. We define a family of models, *Eventual Majority-from-Majority*, $\Diamond$MFM(m), where, roughly speaking, a majority of the processes have timely incoming links from a majority, and the rest have timely incoming links from a minority. In order to strengthen the lower bound, we add a host of additional assumptions (see Table 1.1, row 5): We require a minority of processes to be $\Diamond n$-sources. We replace j-destination assumptions with *j-accessibility* [1, 43], i.e., the existence of bidirectional timely links with j correct processes. Finally, we require reliable links. We show that in the resulting models, $\Diamond$MFM(m), global decision cannot be achieved in bounded time from GSR. Interestingly, these models are strictly stronger than those of [2, 3, 43], which were used for solving consensus. We note, though, that in [43], local decision (of the leader and its accessible destinations) is possible in constant time, whereas in [2, 3], local decision time is unbounded as well.

Finally, we investigate which model is most appropriate in practice. We model link failure probabilities as Independent and Identically Distributed (IID) Bernoulli random variables. Link failure probability is the probability that the link fails to deliver messages by a chosen timeout. We show that the performance of the optimal ES-based algorithm (measured by the expected number of rounds until global decision) deteriorates quickly as we increase the number of processes (n) or decrease the probability that messages arrive on time, and that both $\Diamond AFM$ and $\Diamond LM$ outperform ES. We show that $\Diamond AFM$ is much more scalable than $\Diamond LM$ and ES. Asymptotically, when $n \to \infty$, the expected number of rounds needed for the $\Diamond AFM$-based algorithm to achieve global consensus decision approaches the constant value of 5 rounds, while the expected number of rounds for ES and $\Diamond LM$ goes to ∞ (although the deterioration of $\Diamond LM$ is much slower). Additionally, we investigate the tradeoff between setting longer timeouts and achieving faster decision in a practical setting, using TCP latencies measured by Cardwell et al. [10], and determine the optimal timeout in this setting.

As Table 1.1 shows, there are still several tantalizing gaps between the known upper and lower bounds in various models. Moreover, many additional models can be explored, e.g., in the middle ground between AFM and MFM. We hope that GIRAF will allow researchers to address many such issues in future work. Chapter 11 provides further discussion of future research directions.

Chapter 2

Related Work

State machine replication [35, 47] is a widely used technique for achieving software fault tolerance in distributed systems. With this approach, all replicas perform operations that update the data in the same order, and thus remain mutually consistent. In order to agree on the order of operations, a *consensus* algorithm [38] is often employed. In a consensus algorithm, all processes propose a value and correct processes eventually decide on one of the proposed values, so that no two correct processes decide differently. An instance of consensus is triggered for each user request or group of user requests [36].

2.1 Eventually Stable (Indulgent) Models

It is often unrealistic to assume a fully synchronous system with known time bounds by which all messages arrive. On the other hand, the fundamental FLP result [27] proves that consensus is not solvable in an asynchronous system even if only one process can crash. It is, therefore, often assumed that the system behaves asynchronously for an unbounded period of time, but eventually reaches a stable period, where some properties (assumptions) are satisfied. Different assumptions were proposed in the literature for the system behavior during stable periods [26, 37, 12, 17].

The strongest assumption is that the system is eventually fully synchronous. This model is called *Eventual Synchrony* (or *ES*) [26]. In a synchronous system, there is a known fixed upper bound Δ on message latency and a known fixed upper bound Φ on the relative speeds of different processes (where $\Phi = 1$ means that the processes are completely synchronized). In an asynchronous system, no fixed bounds Δ and Φ exist. In one version of partial synchrony introduced by [26], these bounds exist, but they are not known a priori. In another version, the bounds are known, but they are only guaranteed to hold starting at some unknown time GST (*Global Stabilization Time*).

An alternative to assuming eventual timing guarantees is to use the abstraction of oracle failure detectors [12]. For example, consensus algorithms that make use of group membership mechanisms [15], assume an eventually perfect failure detector, $\Diamond P$ [12]. This failure detector outputs a list of suspected processes at each process, so that eventually correct processes do not suspect any correct process and suspect all faulty processes. Other algorithms use strictly weaker failure detectors, such as the *eventual leader* Ω, which was shown to be the weakest for consensus [11]. This failure detector chooses a *leader* at each process, so that eventually all correct processes choose the same non-faulty leader forever.

Recently, many works have dealt with weakening ES with regard to the timeliness guarantees needed to implement the Ω failure detector abstraction. A simple implementation might periodically send keep-alive messages from all to all, and let each process remember the group of processes from which it heard from in the last broadcast period. Each process elects as leader the process with the lowest process id from this group. This approach requires ES with a known bound on message latency. If time bounds are unknown, a common approach is to increase timeouts until the unknown bound is reached [39]. Aguilera et al. relax the requirement that all communication links need to be eventually timely, to a model in which there is a correct process that eventually maintains timely links with all other processes [1], and to a model in which there is a correct process that eventually has outgoing timely links to all other processes [2]. This process is called a $\Diamond$-source. It was further shown in a recent line of work, that in order to implement Ω, communication with only part of the processes is needed. The two main approaches are listed below.

The work of [32] introduces the notion of a Γ-accurate failure detector. This failure detector's accuracy property is limited by a parameter (Γ). For example, *eventual weak Γ-accuracy* holds if eventually some correct process is never suspected by any correct process in Γ (a set of processes), and the class $\Diamond S(\Gamma)$ consists of failure detectors that satisfy the traditional strong completeness [12] and eventual weak Γ-accuracy. The previously defined $\Diamond S$ failure detector [12] is therefore the same as $\Diamond S(\Gamma)$ where Γ is the group of all processes. Such a failure detector can be easily implemented using keep-alive messages (as was explained earlier) if there is a correct process that has timely outgoing links to the processes in Γ. $\Diamond S(\Gamma)$ can be transformed to $\Diamond S$ [4] and then finally to Ω [16].

Aguilera et al. [3] take a more direct approach. They define a $\Diamond t$-source to be a process that has eventual timely outgoing links to a set of t other processes, and requires this set to be fixed (for a given run). We use similar notations in this research. Any of the t recipients might be faulty. Assuming that t is the upper bound on crash failures, they present an algorithm for Ω in the presence of a single $\Diamond t$-source. In order to implement Ω, both approaches [32] and [3] require the existence of one correct process that has outgoing links to t other, possibly faulty processes. The work of Malkhi et al. [43] requires the existence of a single correct $\Diamond t$-accessible process, i.e. eventually having bidirectional links with t other processes, the t recipients to be correct, but unlike the group of Γ processes in the approach of [32] and the group of t recepients in [3], the t processes of [43] might vary over time. In Chapter 9, we prove that the assumptions of the models in [2, 3, 43] are too weak to allow a bound on global decision time (measuring from system stabilization time) to exist. It should be noted that in [43] local decision, i.e., having at least one process reach a decision, can be achieved in bounded time, whereas in [2, 3] local decision time is unbounded as well. The models we assume in this research are stronger (but all still weaker than ES). In return we present algorithms that achieve global consensus decision in not only bounded time, but constant time, after system stabilization.

2.2 Round-Based Abstractions for Systems with Asynchronous Communication

The original DLS [26] consensus algorithm for the eventually synchronous model uses the idea of studying a general round-based algorithm and separately complementing it with specific implementations of the inter-process coordination needed to implement these rounds in different models. Since DLS deals with eventual synchrony, the achieved coordination must assure that in each round, each process receives a message from all other correct processes. DLS separately deals with synchronization of rounds in different models. For the case of a model with synchronous processes and eventually synchronous communication (where Δ, the bound on message latency, eventually holds), they choose each round to be $n + \Delta$ clock ticks (sending a message to all processes takes n clock ticks, and then Δ clock ticks are used for the receive operation). Additionally, they investigate more difficult models, where the processes and the communication are partially synchronized. Here processes still determine the current round according to their local clock, but the local clock is sometimes adjusted to catch-up with other processes. In these cases, more time is usually needed to simulate each round.

Gafni [28], realized that having a framework that allows different timeliness and failure detector assumptions to be expressed, eases the design and analysis of algorithms using these assumptions. In his framework, rounds of message exchange are employed, where in each round, every process sends a message to every other process. While executing the receive operation, the process consults its Round-by-Round Fault Detector (RRFD). For a given round k, the RRFD outputs a set of possibly faulty processes $D(i, k)$ at every process p_i, and p_i receives message $\langle m, k\rangle$ at round k from every process $p_j \in (\Pi - D(i, k))$. The rounds are communication closed - if a message sent in round k arrives in a later round, it is discarded. Environments are specified using *round-based properties*. A concrete environment model can be completely defined by predicates on the the set $D(i, k)$.

The RRFD paper [28] does not clearly specify the allowed interleaving between rounds of different processes. Although Dutta and Guerraoui [23] use RRFD and interpret it to require that rounds be synchronized among processes, we believe that a more practical interpretation is to assume that rounds of different processes need not be synchronized, as is the case in GIRAF. Considering this interpretation, RRFD can be seen as a special case of GIRAF with the following two restrictions: (i) p_i receives messages from all unsuspected processes in each round: $\forall k \in N \forall p_j \in \Pi - oracle_i(k)$, the link from p_j to p_i is timely in round k, and (ii) communication closed rounds: the action *receive*$(\langle m, k\rangle)_{i,j}$ can only occur in p_i in round k. Note that GIRAF is more general than RRFD. First, the range of the failure detector output in GIRAF can be arbitrary, unlike in RRFD, where it is always a subset of suspected processes. Thus, GIRAF can express models that cannot be expressed in RRFD, e.g., it allows to express Ω and majority, which is impossible in RRFD, as explained in Section 1.2. Second, unlike RRFD, GIRAF models do not have to be communication closed. For example, this allows us to express models that assume reliable albeit untimely links, which is not possible in RRFD. Finally, we prove (Chapter 5) that GIRAF is general enough to capture any oracle-based asynchronous algorithm in the model of [11].

GIRAF also allows us to addresses several open problems, and to revisit the notion of model reducibility defined by [12, 11] to take the time (round) complexity into account, and thus give a better notion of similarity between models. Most of the reductions used previously in distributed systems, did not take time in to account. A few reductions that do take time into account were suggested, e.g., [42]. Our definition is more abstract and is based on the notion of rounds rather than actual time, and thus is more convenient to use.

2.3 Performance after Stabilization

Algorithms and models that tolerate an unbounded period of asynchrony are called indulgent [30]. It was shown [29] that an indulgent algorithm that solves consensus also solves uniform consensus, a variant of consensus in which no two processes (whether correct or faulty) are allowed to decide differently. In this research we are interested in indulgent models, and therefore in solving uniform consensus. In the synchronous model, the tight bound on the number of message exchange rounds for global decision of a uniform consensus algorithm is $t + 2$ [34, 13]. In the eventually synchronous model, there obviously cannot be a bound on the number of rounds needed for global decision, since the system can be asynchronous for an arbitrary long period of time. Several approaches were taken to define this bound differently. The first approach was dedicated to understanding the performance of asynchronous algorithms in runs that are synchronous (or the failure detector is perfect) from the outset [33, 22, 31, 25, 23], typically focusing on the case that all failures are initial. This corresponds to GSR$= 0$ in our model.

The round-based algorithm in DLS for the crash-failure model with $n \geq 2t + 1$ progresses in rounds of message exchange and uses the rotating coordinator approach. Each time a different coordinator is given 4 rounds to achieve a global decision. These 4 rounds are called a *phase*. The algorithm achieves global consensus by round $GST + 4(n + 1)$, and since each round is chosen to be $n + \Delta$ clock ticks, it will take at most $GST + 4(n + 1)(n + \Delta)$ time. In this model, since GST might occur in the middle of a round, it might take up to $n + \Delta$ time to reach GSR from GST. A closer look on the presented protocol shows that it achieves a global decision by round $GSR + 2 + 4(t + 1)$: GSR might occur in the second round of a phase and thus waste that phase (3 rounds are wasted), and the next t phases might be wasted if they belong to faulty coordinators ($4t$ rounds are wasted) and then 4 additional rounds are needed to reach the decision. Since DLS do not assume (as we do) that there are no process failures after GST, $O(t)$ is optimal to within a constant factor since t+2 rounds are needed for uniform consensus even if processors and communication are synchronous and crash failures are assumed [34, 13]. In general, as mentioned by its authors, DLS does not put emphasis on the most efficient way to implement consensus in their models, and leave the optimizations as a direction for future work.

Only very recently, the issue of performance following asynchronous periods has begun to get attention [21, 24]. As noted above, [21] shows that $GSR+2$ is a tight bound for global decision in ES. It uses Gafni's RRFD [28] framework. In [24], Dutta et al. focus on actual time rather than rounds, again in ES; they present an algorithm that decides by $GST + 17\delta$, where δ is a bound on message delay from GST onward (but give no matching lower bound). This result gives a more accurate assessment of the actual running time after GST than our round-count offers. Nevertheless, a similar assessment might be obtained in our model if one can quantify the time it takes the environment's synchronization to establish GSR after GST; this is an interesting subject for future study. We believe that the clean separation we offer between round synchronization and the consensus algorithm's logic allows for more abstract and easier to understand protocol formulations and complexity analyses, as well as for proving lower bounds.

The only previous algorithm presented in the $\Diamond LM$ model, Paxos [37], may require a linear number of rounds after GSR [20]. Most other Ω-based protocols, e.g., [22, 31], wait for messages from a majority in each round (including before GSR), which is undesirable, as it may cause processes to be in arbitrarily unsynchronized rounds when some process first reaches round GSR, causing GSR itself to take a long time. For example, suppose that a model requires that from round GSR onward each process receives the round's message from all correct processes (e.g., ES), and in a given run GSR $= 1000$ and some processes reached round 1000 by GST, but other processes are only in round 3. If no requirements are made before GSR, processes could just skip to round 1000 without further communication. But if the algorithm requires a process to receive messages (e.g., from a majority) in each round even before GSR, they might have to exchange messages many times until they reach round 1000, and only then send round 1000 messages, which are needed to complete round GSR. Dutta et al. [20] allow processes to "skip" rounds in order to re-synchronize in such situations. Implementing such an approach in our framework yields an algorithm that requires one more round than our leader based algorithm, presented in Chapter 6.

Several works, such as [45, 8], provided consensus algorithms for models with different failure types using translations of algorithms for the crash-failure model. These works deal with the synchronous communication model, whereas in this thesis we deal with models that are weaker than the ES model. The closest analogy to the type of failures we deal with might be the general omission model (both receive and send omissions). As proved by [45], simulating the crash failure model is possible if $n \geq 2t + 1$ where t is the number of processes that experience the general omission failures. Let us observe the $\Diamond AFM$ model as an example. Even if we consider GSR to be 0, i.e. a synchronous and not ES system, we cannot assure that only a fixed group of minority of processes experience the failures - each process might lose part of its outgoing messages. Using these methods of translation as an alternative to a direct algorithm, such as the one suggested by us for the $\Diamond AFM$ model in Chapter 8, is therefore not possible.

Chapter 3

Model and Problem Definition

3.1 Distributed Computation Model

We consider an asynchronous distributed system consisting of a set Π of $n > 1$ processes, $p_1, p_2, \ldots, p_n$, fully connected by communication links. Processes and links are modeled as deterministic state-machines, called I/O automata [41].

Formally, an I/O automaton is defined as the following four-tuple[1]: a signature, composed of *actions*, which are classified as *input*, *output*, and *internal*; a set of states; a set of initial states (note that the initial state is not unique), and a transition function - from the cross-product between states and actions, to states. The transitions are triggered by actions. An automaton's interface is determined by its input and output actions, which are collectively called *external actions*.

An action π of automaton A is *enabled* in state *s* if A has a transition of the form (s, π, s'). The transitions triggered by input actions are always enabled, whereas those triggered by output and internal actions (collectively called *locally controlled* actions), are preconditioned on the automaton's current state. Since every input action is required to be enabled in every state, automata are said to be *input-enabled*. This assumption means that the automaton is not able to somehow "block" input actions from occurring.

A run of I/O automaton A is a sequence of alternating states and actions $s_0, \pi_1, s_1, \ldots$, where s_0 is an initial state of A (from A's set of initial states), and each triple (s_{i-1}, π_i, s_i)

[1]The definition of [41] includes a fifth component, called *tasks*. This notion in not used in this thesis.

is a transition of A. The *trace* of a run α of A is the subsequence of α consisting of the external actions in α. An infinite run α of A is *fair* if every locally controlled action of A either occurs infinitely often in α or is disabled infinitely often in α. A fair trace of A is the trace of a fair run of A. An automatons external behavior is specified in terms of the properties of its traces (e.g., the properties specified in Section 3.2). Liveness properties are required to hold only in fair traces. We only consider infinite fair runs in this thesis.

Action a of process p_i is denoted a_i. A process p_i interacts with its incoming link from process p_j via the *receive(*m*)*$_{i,j}$ action, and with its outgoing link to p_j via the *send(*m*)*$_{i,j}$ action. Communication links do not create, duplicate, or alter messages (this property is called *integrity*). Messages may be lost by links.

A threshold of t of the processes may fail by crashing. The failure of process p_i is modeled using the action *crash*$_i$, which disables all locally controlled actions of p_i. A process that does not fail is *correct*. The actual number of failures occurring in a run is denoted f. Process p_i is equipped with a *failure detector oracle*, which can have an arbitrary output range [11], and is queried using the *oracle*$_i$ function.

3.2 GIRAF – General Round-Based Algorithm Framework

Figure 3.1 presents GIRAF, a generic round-based distributed algorithm framework. To implement a specific algorithm, GIRAF is instantiated with two functions: *initialize()*, and *compute()*. Both are passed the oracle output, and *compute()* also takes as parameters the set of messages received so far and the round number. These functions are non-blocking, i.e. they are not allowed to wait for any other event.

Each process's computation proceeds in *rounds*. The advancement of rounds is controlled by the environment via the *end-of-round* input action. The *end-of-round*$_i$ actions occur separately in each process p_i, and there are no restrictions on the relative rate at which they occur at different processes, i.e., rounds are not necessarily synchronized among processes. The *end-of-round* action first occurs in round 0, whereupon it queries the oracle and calls *initialize()*, which creates the message for sending in the first round

```
States:
    $k_i \in N$, initially 0    /*round number*/
    $sent_i[\Pi] \in$ Boolean array, initially $\forall p_j \in \Pi : sent_i[j] = true$
    $FD_i \in$ OracleRange, initially arbitrary
    $M_i[N][\Pi] \in Messages \cup \{\bot\}$, initially $\forall k \in N \forall p_j \in \Pi : M_i[k][j] = \bot$

Actions and Transitions:
    input $receive(\langle m, k\rangle)_{i,j}$, $k \in N$
        Effect: $M_i[k][j] \leftarrow m$

    input $end\text{-}of\text{-}round_i$
        Effect: $FD_i \leftarrow oracle_i\ (k_i)$
                if $(k_i = 0)$ then $M_i[1][i] \leftarrow initialize\ (FD_i)$
                else $M_i[k_i + 1][i] \leftarrow compute\ (k_i, M_i, FD_i)$
                $k_i \leftarrow k_i + 1$
                $\forall p_j \in \Pi : sent_i[j] \leftarrow false$

    output $send\ (\langle M_i[k_i][i], k_i\rangle)_{i,j}$
        Precondition: $sent_i[j] = false$
        Effect: $sent_i[j] \leftarrow true$
```

Figure 3.1: GIRAF: Generic algorithm for process p_i (I/O automaton).

(round one). Subsequently, during each round, the process sends a message to all processes and receives messages available on incoming links, until the *end-of-round* action occurs, at which point the oracle is queried and *compute()* is called, which returns the message for the next round. We say that an event of process p_i *occurs in round* k of run r, if there are exactly k invocations of *end-of-round*$_i$ (i.e., *end-of-round* invocations at process p_i), before that event in r.

For simplicity, we have the algorithm send the same message to all processes in each round; this is without loss of generality as we are not interested in message complexity as a performance metric. The outgoing message is stored in the incoming message buffer, $M_i[k_i+1][i]$, hence self-delivery is ensured. The environment might decide not to send the message of a round to any subset of processes, i.e., it might invoke *end-of-round*$_i$ in round k without a *send*$(m)_{i,j}$ action ever happening in round k for a process p_j. However, some of our environment definitions below will restrict this behavior and require messages to be sent. In any case, self-delivery is always preserved.

Our framework can capture any asynchronous oracle-based message-passing algorithm in the general model of [11] (see Chapter 5). Thus, GIRAF does not restrict the allowed algorithms in any way, but rather imposes a round structure that allows for analyzing them.

Each run is determined by the algorithm automaton's state transitions, and the *environment's* actions, consisting of (i) scheduling *end-of-round* actions; (ii) oracle outputs; and (iii) *send* and *receive* actions of the communication links. Environments are specified using *round properties*, which are predicates evaluated on a run, that restrict the oracle outputs or message arrivals in each round. Formally, a round property is a predicate $\phi(k, r)$ of round number k in run r. For simplicity of notation, we omit the r, and write $\phi(k)$. For example, "the link from p_s to p_d is timely in round k" is a round property. We consider two types of environment properties: *perpetual* properties, which hold in each round, and *eventual* properties, which hold from some (unknown) round onward. A perpetual property P is a property of the form: $(\forall l \geq 0)(\phi(l))$, where $\phi(l)$ is a round property. For P as above, the eventual property $\Diamond P$ is defined as: $(\exists l \geq 0)(\forall l' \geq l)(\phi(l'))$. In every run r of environment model M, there exists a round GSR$(r)_M$, from which onward there are no failures, and all eventual properties of M hold. Formally:

Definition (GSR$(r)_M$): Let M be a model. In each run r in M, GSR$(r)_M$ is the first round k, s.t. $\forall$ round $k' \geq k$ no process fails, and for every eventual property $\Diamond P = (\exists l \geq 0)(\forall l' \geq l)(\phi(l'))$ in M, $\phi(k')$ holds.

As was defined earlier, the first communication round is round one. By slight abuse of terminology, we say that GSR$(r)_M = 0$ in a run r if (i) there are no failures in r; (ii) the oracle properties of M (if defined) hold from round zero in r; and (iii) the communication properties of M hold from round one in r. (We henceforth omit the M and (r) where it is clear from the context)

Note that although, in general, rounds are not synchronized among processes, we specify certain environment properties that do require some synchronization, e.g., that some messages are received at one process at the same round in which they are sent by another. Therefore, an implementation of an environment that guarantees such properties needs to employ some sort of round or clock synchronization mechanism (e.g. [26, 5], or using GPS clocks).

3.3 Environment Properties

We define several environment properties in GIRAF, mostly in perpetual form. Prefixing a property with $\Diamond$ means that it holds from GSR onward.

Communication Properties Every process has a "link" with itself, and though it is not an actual physical link, it counts toward the j timely links in the definitions below. Some of the properties that require j timely links may appear with a subscript v (variable), which indicates that the set of j timely links is allowed to change in each round. Note that link integrity is assumed by the model. When characterizing a link, we denote the source process of the link by p_s, and the recipient by p_d.

reliable link: $\forall k \in N^+$ if *end-of-round*$_s$ occurs in round k and p_d is correct, then p_d receives the round k message of p_s.

timely link in round k**:** if *end-of-round*$_s$ occurs in round k and p_d is correct, then p_d receives the round k message of p_s, in round k.

Synchrony: $\forall k \in N^+$ all links are timely in round k.

Eventual Synchrony (ES): $\Diamond$Synchrony.

j**-source:** process p is a j*-source* if there are j processes to which it has timely outgoing links in every round; p is a j*-source*$_v$ if in every round it has j timely outgoing links. (Correctness is not required from the recipients.)

j**-destination:** correct process p is a j*-destination* if there are j correct processes from which p has timely incoming links in every round; p is a j*-destination*$_v$ if it has j timely incoming links from correct processes in every round.

j**-accessible:** correct process p is j*-accessible* if there are j correct processes with which p has timely bidirectional links in every round. (We do not consider variable j-accessibility in this thesis.)

Note that the reliable and timely link properties assure that the environment sends messages on the link, i.e., the *end-of-round*$_s$ action in round k is preceded by a *send*$(m)_{s,d}$ action in round k.

Failure Detector Properties We next define several oracle properties [11, 12]. The range of the *oracle()* function for S (and $\Diamond S$) is 2^{Π} – a group of suspected processes. For *leader* (and Ω), the range is Π.

S **failure detector:** $\Diamond SC$ (strong completeness) – eventually every faulty process is suspected by every correct process, and WA (weak accuracy) – some correct process is not suspected.

leader: $\exists$ correct p_i s.t. for every round $k \in N$ and every $p_j \in \Pi$, $oracle_j(k) = i$.

Ω **failure detector:** $\Diamond$ *leader*.

3.4 Consensus and Global Decision

A consensus problem is defined for a given value domain, *Values*. In this thesis, we assume that *Values* is a totally ordered set. In a consensus algorithm, every process p_i has a read-only variable $prop_i \in$ *Values* and a write-once variable $dec_i \in$ *Values* $\cup\{\perp\}$. In every run r, $prop_i$ is initialized to some value $v \in$ *Values*, and dec_i is initialized to $\perp$. We say that p_i *decides* $d \in$*Values* in round k of r if p_i writes d to dec_i when $k_i = k$ in r.

An algorithm A solves consensus if in every run r of A the following three properties are satisfied: (a) (*validity*) if a process decides v then $prop_i = v$ for some process p_i, (b) (*agreement*) no two correct processes decide differently, and (c) (*termination*) every correct process eventually decides.

We say that a run of A achieves *global decision* at round k if (1) every process that decides in that run decides at round k or at a lower round; and (2) at least one process decides at round k.

Chapter 4

Complexity of Reductions

In discussing different models, the question of *reducibility* naturally arises – one is often interested in whether one model is stronger than another, or how “close” two models are. The classical notion of reducibility among models/oracles [12, 11] does not take complexity into account. We use GIRAF to provide a more fine-grained notion of similarity between models.

We first explain how classical reducibility is expressed for GIRAF models. Reducibility (in the “classical” sense) means that one model can be emulated in another. A simulation from a GIRAF model M_1 to another (GIRAF or non-GIRAF) model M_2, must work within the *initialize()* and *compute()* functions in M_1, which must be non-blocking. Simulating a GIRAF model M_2 means invoking the *initialize*$_A$*()* and *compute*$_A$*()* functions of some algorithm A that works in M_2, while satisfying the properties of M_2. In particular, if M_1 and M_2 are both GIRAF models, then a reduction algorithm $T_{M_1 \to M_2}$ instantiates the *initialize()* and *compute()* functions, denoted *initialize*$_T$*()* and *compute*$_T$*()*, and invokes *initialize*$_A$*()* and *compute*$_A$*()* in model M_1. If algorithm $T_{M_1 \to M_2}$ exists, we say that M_2 is reducible to M_1 (or weaker than M_1), and denote this by $M_1 \geq M_2$. M_1 is equivalent to M_2 if $M_1 \geq M_2$ and $M_2 \geq M_1$.

We next extend the notion of reducibility, and introduce
α-reducibility, which takes the reduction time (round) complexity into account. Note that the definition of a run’s GSR is model-specific: GSR$(r) = k$ in model M if k is the first round from which onward no process fails and the eventual properties *of* M are satisfied.

We denote GSR in model M and run r by $\text{GSR}_M(r)$.

Definition (α-reducibility). *For $\alpha : \mathrm{N} \rightarrow \mathrm{N}$, we say that model M_2 is α-reducible to model M_1, denoted $M_1 \geq_\alpha M_2$, if there exists a reduction algorithm $T_{M_1 \rightarrow M_2}$ s.t. for every run r and every $l \in \mathrm{N}$, round $GSR_{M_2}(r) + l$ in model M_2 occurs at most in round $GSR_{M_1}(r) + \alpha(l)$ in model M_1.*

Definition (k-round reducibility). *Model M_2 is k-round reducible ($k \in N$) to model M_1, denoted $M_1 \geq_k M_2$, if $M_1 \geq_\alpha M_2$ s.t. $\alpha(l) = l + k$.*

In particular, if $M_1 \geq_0 M_2$ then model M_2 can be simulated in model M_1 with no performance penalty. In Chapter 7 we use the notion of k-round reducibility to prove that $\Diamond S$ is 0-round reducible to $\Diamond n$-source.

Chapter 5

Generality of GIRAF

In this chapter we show how GIRAF relates to the framework of [11]. A computation step in the model of [11] consists of (i) receiving a message, (ii) consulting the oracle, (iii) using the process's algorithm ($A(p)$ in their notation) to perform local computation and generate an outgoing message; and (iv) sending the message. Moreover, reliable links are assumed.

Lemma 5.1. *Every model M_1 in the framework of [11] is equivalent to a GIRAF model M_2, where the only environment properties are the same oracle properties as in M_1 and reliable links.*

Proof. We prove that the framework of [11] with model M_1 can be used to implement the environment for GIRAF resulting in model M_2. Note that in [11], $A(p)$ is invoked upon every message receipt after the oracle is queried, and the oracle output and incoming message are available to it. To run the generic GIRAF algorithm in model M_1, we have $A(p)$ first invoke *initialize()*, and subsequently invoke *compute()* every time it is called to take a step. $A(p)$ passes to these functions the oracle output. To *compute()*, it also passes the set of messages received thus far, and a counter of the number of times *compute()* is called. *compute()* or *initialize()* returns a message, which is sent immediately afterwards. Every perpetual property guaranteed by the oracle of M_1 holds starting from the first round in M_2, and every eventual property of the oracle is eventually true in M_2, guaranteeing that the properties of M_2's oracle are preserved.

We next prove that GIRAF with model M_2 can be used to simulate the framework of [11]. Given an algorithm $A(p)$ designed for the framework of [11] and model M_1, we make the *compute()* function of GIRAF invoke a series of steps of $A(p)$ – one invocation for every message m added to M since the previous time *compute()* was activated. *compute()* then aggregates all the messages that these steps return into one composite message which is returned to GIRAF generic algorithm (to be sent in the next round). When a step of $A(p)$ queries the oracle, it is given the oracle output passed to *compute()*. Note that each step of $A(p)$ does not wait for anything, and therefore *compute()* is non-blocking, as required by GIRAF. Since the message arrival order is arbitrary in [11], this a valid run in M_1. Every perpetual property of M_2's oracle is preserved starting from the first round and is therefore true starting from the first activation of $A(p)$, and every eventual property will hold starting from GSR in M_2 and therefore eventually holds in M_1. $\square$

Suppose we have an algorithm A(p) running in the framework of [11]. The above lemma shows how to run the same algorithm in GIRAF. An interesting question to ask is whether by running it in GIRAF, its performance was compromised. To address performance of A(p) in the CHT framework, we use a metric suggested by [33] for indulgent algorithms: the number of communication steps in failure free synchronous runs. This metric was used in numerous papers dealing with CHT (and other) models. Recall that by definition, in synchronous runs, processes execute in synchronous communication steps (also called rounds). In each step, every process can send messages to any number of other processes. In CHT, processes send messages to every other process. For the purpose of this analysis, we assume that each message takes exactly Δ time units, and that the local computation takes 0 time.

Let us observe a failure free synchronous run of A(p). By the end of each communication step, every process p_i receives a message from all processes. It then invokes A(p) once for every received message. Assuming that the execution of $A(p)$ takes 0 time and the communication step takes Δ, the round will take Δ time units. When we run A(p) in GIRAF, using the simulation described in Lemma 5.1 above, Δ time units after the start of a round, the requirements of the synchronous model are satisfied, and thus *end-of-round*$_i$ is called at every process p_i. The execution of the callback for *end-of-round*$_i$ takes 0 time - it simply queries the oracle once and then executes $compute()$ (or $initialize()$) which in turn call A(p) for each message received in the current round. Therefore, each round will take Δ time units as well. We therefore have the algorithm proceed in rounds that take the same time as the rounds in the CHT framework, and will thus take the same time to finish its task. We conclude, that running the algorithm in GIRAF does not incur a penalty according to this metric.

Note that even if an indulgent algorithm does not have every process send in every step to all processes, having every process send to all and wait for messages from all every Δ time does not hamper performance in synchronous runs, since in these runs every message takes Δ time to arrive.

Chapter 6

Optimal Leader-Based Algorithm in $\Diamond LM$

The $\Diamond$LM model requires that each process have a majority of timely incoming links (from GSR onward), which can vary in each round, and an Ω oracle that selects a correct $\Diamond n$-source as leader. Formally:

$\Diamond LM$ *(Leader-Majority)*: $t < n/2$, Ω failure detector, the leader is a $\Diamond n$-source, and every correct process is a $\Diamond(\left\lfloor \frac{n}{2} \right\rfloor + 1)$-destination$_v$.

The $\Diamond$LM model is strictly weaker than ES: it is easy to show that $ES \geq_0 \Diamond LM$, i.e. to simulate $\Diamond LM$ in ES with no penalty on GSR, since $\Diamond LM$ requires less $\Diamond$timely links than ES, and the Ω failure detector output in any given round k can be (for example) the lowest-id of a process whose round k message was received in round k. Starting from GSR_{ES} this process is assured to be correct and since in ES all correct processes receive the same set of messages in each round, all simulated oracles will believe in the same correct process starting at round GSR_{ES}, meaning that $GSR_{\Diamond LM} = GSR_{ES}$.

6.1 Algorithm

Figure 6.1 presents a leader-based consensus algorithm for $\Diamond$LM, which reaches global decision by round GSR+2. In runs with GSR $= 0$, this means that consensus is achieved in 2 rounds, which is tight [13, 33]. In runs with GSR > 0, global decision is reached in

```
**Additional state**
    $est_i \in Values$, initially $prop_i$
    $ts_i$, $maxTS_i$, $lastApproval_i \in N$, initially 0
    $prevLD_i$, $newLD_i \in \Pi$
    $msgType_i \in \{$PREPARE, COMMIT, DECIDE$\}$, initially PREPARE
**Message format**
    $\langle msgType \in \{$PREPARE, COMMIT, DECIDE$\}$, $est \in Values$, $ts \in N$, $leader \in \Pi$, $lastApproval_i \in N\rangle$
**procedure** initialize($leader_i$)
    $prevLD_i \leftarrow newLD_i \leftarrow leader_i$
    return message $\langle msgType_i, est_i, ts_i, newLD_i, lastApproval_i\rangle$   /*round 1 message*/
**procedure** compute($k_i$, M[*][*], $leader_i$)
    **if** $dec_i = \bot$ **then**
                /*Update variables*/
        $prevLD_i \leftarrow newLD_i$; $newLD_i \leftarrow leader_i$
        $maxTS_i \leftarrow \max\{ m.ts \mid m \in M[k_i][*] \}$
        **if** $|\{ j \mid M[k_i][j] \neq \bot \}| > \lfloor n/2 \rfloor$ **then**
            $lastApproval_i \leftarrow k_i$
                /*Round Actions*/
        **if** $\exists m \in M[k_i][*]$ s.t. $m.msgType =$ DECIDE **then**   /*decide-1*/
            $dec_i \leftarrow est_i \leftarrow m.est$; $msgType_i \leftarrow$ DECIDE
        **else if** ($|\{ j \mid M[k_i][j].msgType =$ COMMIT $\}| > \lfloor n/2 \rfloor$)
              **and** ($M[k_i][prevLD_i].msgType = M[k_i][i].msgType =$ COMMIT) **then**   /*decide-2*/
            $dec_i \leftarrow est_i$; $msgType_i \leftarrow$ DECIDE
        **else if** ($|\{ j \mid M[k_i][j].leader = prevLD_i \}| > \lfloor n/2 \rfloor$)   /*commit-1*/
                  **and** ($M[k_i][prevLD_i].lastApproval = k_i - 1 \wedge M[k_i][prevLD_i].leader = prevLD_i$)   /*commit-2*/
              **and** ($newLD_i = prevLD_i$) **then**   /*commit-3*/
            $est_i \leftarrow M[k_i][prevLD_i].est$; $ts_i \leftarrow k_i$; $msgType_i \leftarrow$ COMMIT;
        **else**
            $est_i \leftarrow$ any $est' \in \{ M[k_i][j].est \mid M[k_i][j].ts = maxTS_i \}$
            $ts_i \leftarrow maxTS_i$; $msgType_i \leftarrow$ PREPARE
    return message $\langle msgType_i, est_i, ts_i, newLD_i, lastApproval_i\rangle$   /*round $k_i + 1$ message*/
```

Figure 6.1: Optimal leader–based algorithm for $\Diamond LM$, code for process p_i.

3 rounds, numbered GSR, GSR+1, and GSR+2, which also matches the lower bound for ES [21].

The algorithm in Figure 6.1 works in GIRAF, and therefore implements only the *initialize()* and *compute()* functions. These function are passed *leader*$_i$, the leader trusted by the oracle.

The main idea of the algorithm, which ensures fast convergence, is to trust the leader even if it competes against a higher bid of another process. In contrast, Paxos [37] initiates a new "ballot", that is, aborts any pending attempts to decide on some value, whenever a higher timestamp is observed, potentially leading to linear running time after GSR [20]. In order to ensure that the leader does not propose a value that contradicts previous agreement, the *lastApproval* variable (and message-field) conveys the "freshness" of the leader's proposed value, and the leader's proposals are not accepted if it is not up-to-date.

We now describe the protocol in more detail. Process p_i maintains the following local variables: an estimate of the decision value, est_i initialized to the proposal value ($prop_i$); the timestamp of the estimated value, ts_i, and the maximal timestamp received in the current round, $maxTS_i$, both initialized to 0; the index of the last round in which p_i receives a message from a majority of processes, *lastApproval*$_i$, initialized to 0; the leader provided by the oracle at the end of the previous round, *prevLD*$_i$, and in the current round, *newLD*$_i$; and the message type, *msgType*$_i$, which is used as follows: If p_i sees a possibility of decision in the next round, then it sends a COMMIT message. Once p_i decides, it sends a DECIDE message in all subsequent rounds. Otherwise, the message type is PREPARE.

We now describe the computation of round k_i. If p_i has not decided, it updates its variables as follows. It saves its previous leader assessment in *prevLD*$_i$, and its new leader (as passed by the oracle) in *newLD*$_i$ (line 14). It stores the highest timestamp received in $maxTS_i$. If p_i receives a message from a majority, it sets *lastApproval*$_i$ to the round number, k_i. It then executes the following conditional statements:

- If p_i receives a DECIDE message then it decides on the received estimate by writing that estimate to dec_i (line 20).

- If p_i receives COMMIT messages from a majority of processes, including itself and its leader, then p_i decides on its own estimate (line 22).

- Let *prevLD*$_i$ be the leader indicated in p_i's round k_i message. Consider the following three conditions (line 23): *commit-1*: p_i receives round k_i messages from a majority of processes that indicate *prevLD*$_i$ as their leader; *commit-2*: p_i receives a message from *prevLD*$_i$ that has *prevLD*$_i$ as the leader, and *lastApproval* set to $k_i - 1$; and *commit-3*: *prevLD*$_i$ = *newLD*$_i$. If all three conditions are satisfied, then p_i sets its message type (for the round $k_i + 1$ message) to COMMIT, adopts the estimate received from *prevLD*$_i$, say est', and sets its timestamp to the current round number k_i (line 24). We say that p_i *commits in round* k_i with estimate est'.

- Otherwise, p_i adopts the estimate and the timestamp of an arbitrary message with the highest timestamp $maxTS_i$, and sets the message type to PREPARE (lines 26–27).

Finally, p_i returns the message for the next round.

Theorem 6.1. *The algorithm solves consensus by round GSR(r) + 2.*

Proof. From Lemma 6.8, every correct process decides by round $GSR(r) + 2$. Validity holds, since the decision can only be one of the initial estimates of the processes. Uniform agreement is proven in Lemma 6.7. □

6.2 Correctness

We first informally explain the correctness of the algorithm. Our main lemma (Lemma 6.7) shows that no two processes decide differently, by showing that if some process decides x in round k, then from round $k - 1$ onward, the only committed estimate is x. (This proves agreement since a decision is made when either a DECIDE or a majority of COMMITs is received.) We now intuitively explain why this is correct. The claim is proven by induction on round number. Let p_i be the first process that decides, and denote its decision value by x, and the decision round by k. (the decision is by rule *decide-2*; rule *decide-1* is not applicable since p_i is the first process to decide). Therefore, in round k, p_i hears COMMIT from majority M, including itself and its round k *prevLD*, p_l, and decides on its own estimate, x. Let us first examine round $k - 1$. Processes of M commit in this round. Rules *commit-1* and *commit-3* ensure that all COMMIT messages sent in this round have the same estimate and leader fields, namely, x, and p_l. Additionally, it is easy to see that a process's timestamp never decreases. Thus, since processes of M commit in round $k - 1$, they have timestamps of at least $k - 1$ in all ensuing rounds. Now consider round k. Any process that commits in round k hears from a majority with the same leader, and since this majority intersects M, the leader is p_l. Therefore, any commitment in round k is made with the estimate of p_l, i.e., x.

We now consider the inductive step, i.e., round $k' > k$. If p_i commits in round k', it commits on the estimate of its leader. If that leader sends a COMMIT message, by induction, its estimate is x. Otherwise, the leader sends a PREPARE message. By *commit-2*, that leader's *lastApproval* field is set to $k' - 1 \geq k$, implying that the leader receives a message from a majority of processes in round $k' - 1$. Therefore, it receives at least one message from a process in M with timestamp at least $k - 1$. Since the highest timestamp received is adopted, the leader adopts timestamp $ts \geq k - 1$ and some estimate z. It is easy to see that if a message (other than DECIDE) is sent with timestamp ts and estimate z, then some process commits z in round ts. Therefore, some process commits z in a round $\geq k - 1$. By induction, we get that $z = x$. Therefore, the leader adopts x with the maximal timestamp in round $k' - 1$, and p_i commits x in round k'.

We now formally prove correctness.

Lemma 6.2. *A process's timestamp at the start of round k is less than k.*

Proof. We prove the claim by induction on the round number k'. Base case: $k' = 1$. The claim is correct since a process's timestamp is initialized to 0. The induction hypothesis is that the claim holds up to round k'. Let us inspect the possible actions of processes at the end of round k'. A process can decide and in this case its timestamp does not change and in round $k' + 1$ it will remain less or equal to $k' - 1$, by the induction hypothesis. Alternatively, a process may commit, and then (on line 24) it will adopt k' as its new timestamp for round $k' + 1$, and the claim holds here as well. Finally, a process may adopt the timestamp of a round k' message it received (lines 26-27) and again, by induction hypothesis, the claim is true. □

Lemma 6.3. *A process's timestamp is non-decreasing.*

Proof. Observe that when a process decides, its timestamp does not change. It does not change in the following rounds as well. If a process p_i does not decide in round k, then it can change its timestamp by adopting either k (when committing on line 24) or the maximum timestamp (of a round k message) it received in round k as its new timestamp (lines 26-27). Since p_i receives its own message in round k, the latter is not lower than its current timestamp. In case it commits, since according to Lemma 6.2, its old timestamp cannot exceed $k-1$, by adopting k it can only increase. □

Lemma 6.4. *For every round k, all processes that commit in round k, have the same est and $newLD$ values.*

Proof. Consider two processes p_i and p_j that commit in round k with estimates est_i and est_j, and $leader$ values $newLD_i$ and $newLD_j$, respectively. Also, in round k, let $prevLD_i$ be the leader of p_i and $prevLD_j$ be the leader of p_j. From *commit-1*, each of them has received in round k a majority of messages that contain $prevLD_i$ and $prevLD_j$ as leader, respectively. As two majorities intersect, $prevLD_i = prevLD_j$. Furthermore, from *commit-3*, $newLD_i = prevLD_i$ and $newLD_j = prevLD_j$. So, $newLD_i = prevLD_i = prevLD_j = newLD_j$. From the algorithm, p_i commits with the estimate sent by $prevLD_i$, and p_j commits with the estimate sent by $prevLD_j$. As $prevLD_i = prevLD_j$, p_i and p_j commit with the same estimate. □

Lemma 6.5. *If some process sends a* PREPARE *or* COMMIT *message with timestamp $ts > 0$ and estimate x then some process commits in round ts with estimate x.*

Proof. We prove the claim by induction on the round number k', starting from a round k_0 in which a message with the timestamp ts is first sent with some estimate x', by some process p_j.

Base Case. $k` = k_0$. From the definition of k_0, p_j does not receive a message with ts from another process in an earlier round. Thus, p_j commits with timestamp ts and estimate x' in round $k_0 - 1$, and by the algorithm, $k_0 - 1 = ts$.

Induction Hypothesis. If any process sends a PREPARE or COMMIT message in round k_1, such that $k_0 \leq k_1 \leq k'$, with timestamp ts and some estimate x'', then some process commits in round ts with estimate x''.

Induction Step. We need to show that if in round $k' + 1$, a process sends a PREPARE or COMMIT message with timestamp ts and some estimate x'' then some process commits in round ts with estimate x''. Observe, that if a COMMIT message is sent, it would have a timestamp equal to the previous round number k', and since $ts = k_0 - 1 < k'$ (by the base case), this case is not possible. Observe that if a PREPARE message is sent in round $k' + 1$ with timestamp ts and estimate x'', the sending process must have adopted the timestamp together with the estimate from some PREPARE or COMMIT message sent in round k'. By the induction hypothesis, we get that some process commits in round ts and estimate x''. □

Please note that the claim in Lemma 6.5 does not hold for DECIDE messages, since a process decides adopting only the estimate and not the associated timestamp from another DECIDE message.

Lemma 6.6. *If a process p_i decides in round k by rule* decide-2 *on estimate x, then every process that commits in round k, commits with estimate x.*

Proof. Suppose for the purpose of contradiction that a process p_j commits with $y \neq x$ in round k. Since p_j does not decide in round k, it evaluates rules *decide-1* and *decide-2* to false. p_j commits the estimate that it receives from its leader (line 24). We denote this leader by p_l. By rule *commit-1*, there is a majority of proccesses that send a round k message with p_l as leader. Let us denote this majority by M_1. Observe process p_i that decides in round k. p_i receives a COMMIT message from a majority of processes, including its leader $prevLD_i$ and itself. We denote this majority by M_2. By Lemma 6.4 every COMMIT message sent by a process in M_2 has the same leader field and the same estimate (x). Since M_1 and M_2 intersect, the leader field indicates p_l. Since p_i receives a COMMIT message from itself, it also sends a round k message with p_l as leader. Since what p_i actually sent is now in his $prevLD_i$ variable, we get that $prevLD_i = p_l$. Since p_i receives a message $M_i[k][prevLD_i]$ with $msgType =$ COMMIT, we conclude that p_l sends a COMMIT message in round k and as was explained, this means that its message includes x as the estimate. This contradicts our assumption that p_j sees an estimate $y \neq x$ sent by p_l. □

Lemma 6.7 (Uniform Agreement)**.** *No two processes decide differently.*

Proof. Let k be the lowest numbered round in which some process decides. Suppose process p_i decides x in round k. Since no process decides in an earlier round, p_i decides by rule *decide-2*. Therefore, p_i receives a majority of COMMIT messages in round k, and it decides on the estimate of one of the COMMIT messages (the one from itself). From Lemma 6.4, all COMMIT messages include the same estimate and leader, say p_l.

Thus p_i receives a round k message of the form $\langle$COMMIT, x, $k-1$, pl, $*\rangle$ from a majority of processes, and hence, a majority of processes commit in round $k-1$ with estimate x. Let us denote this majority of processes by S_x. Note that $k-1 \geq 1$ since according to the pseudo-code, the first round of the algorithm is round number 1. We claim that if any process commits or decides in round $k' \geq k-1$ then it commits or decides x. The proof is by induction on round number k'.

Base Case. $k' = k-1$. As processes in S_x commit x in round $k-1$, from Lemma 6.4, no process commits with an estimate different from x in round $k-1$. By the definition of k, no process decides in round $k-1$.

Induction Hypothesis. If any process commits or decides in any round $k1$ such that $k-1 \leq k1 \leq k'$, then it commits with estimate x or decides x.

Induction Step. decision in round $k'+1$. If some process p decides in round $k'+1$, then in that round either some other process sends a DECIDE message with decision value y or p sends a COMMIT message with estimate y. In both cases, by the induction hypothesis, $y = x$.

commit in round $k'+1$. Suppose by contradiction that some process p_j commits in round $k'+1$ with estimate $z \neq x$. First, since p_i decides by rule *decide-2* in round k, by Lemma 6.6 we have that $k'+1 \neq k$. Since we know by the induction hypothesis that $k' \geq k-1$ we now get that $k' > k-1$. Since p_j commits, it does not receive any DECIDE message in round $k'+1$. Since *commit-2* evaluated to true for p_j, a message $m = \langle type\ (\neq DECIDE), z, ts_z, ld, k'\rangle$ was received by p_j in round $k'+1$ from the leader ld. Notice that ts_z might be different than $maxTS_i$ of round $k'+1$.

Observe the $lastApproval$ field of the message m. Its value is k'. Since $k' > k-1 \geq 1$ we get that $k' > 1$. Since the $lastApproval$ field can become greater than 0 only on line 17 of the $compute()$ function, this indicates that the leader received a message from a majority of processes in round k', and therefore it must have heard from at least one process $p_a \in S_x$. Recall that every process in S_x commits in round $k-1$ with estimate x. Thus p_a has timestamp $k-1$ at the end of round $k-1$. From Lemma 6.3, since $k' > k-1$, p_a's timestamp is at least $k-1$.

If *type* =COMMIT, this means that $ts_z = k'$ (line 24). As was explained, $k' > 1$, and by Lemma 6.5 we get that some process commits in round k' with estimate $z \neq x$. This is a contradiction to the induction hypothesis. If *type* = PREPARE, it means that ts_z is the maximum timestamp the leader received in any message of round k' (lines 26-27). Because it received a message from p_a and because, according to Lemma 6.2, the highest timestamp that can be received in round $k'+1$ is k', we get that $k-1 \leq ts_z \leq k'$, and since (by Lemma 6.5) there must be a process that commits in round ts_z with estimate $z \neq x$ (recall that $k-1 > 0$), this is a contradiction to the induction hypothesis. □

6.3 Performance

We first give an intuitive explanation why in round GSR+1, every correct process p_i that does not decide by the end of that round evaluates the three *commit* rules (line 23) to *true*. Since p_i does not decide by the end of GSR+1, all the processes it hears from in this round do not decide by round GSR. By definition of $\Diamond LM$, from round GSR onward, each correct process receives messages from a majority of correct processes, including its leader, p_l. Therefore, the *lastApproval* field of every round GSR+1 message is GSR (notice for the case of GSR= 0 that *lastApproval* is initialized to 0). Moreover, it is assured by the Ω failure detector, that from round GSR onward, all processes trust the same leader, p_l. Therefore, from round GSR+1 onward, all running processes (including the leader p_l) send the same leader identifier in their messages. (Note that rule *commit-3* is assured to be true only starting at round GSR+1, since *prevLD*$_i$ of round k_i = GSR is based on the oracle's output in round GSR−1, in which it is not assured that all processes trust the same leader.) We conclude that in round GSR+2 every correct process sends a COMMIT or DECIDE message, and by the end of that round, every correct process decides.

We now give a formal proof.

Lemma 6.8. *In every run r, all correct processes decide by round $GSR(r) + 2$.*

Proof. Observe that in our model every correct process executes an infinite number of rounds, and in particular, executes round $GSR(r) + 2$. We prove the lemma by contradiction. Assume that some correct process p_j does not decide by round $GSR(r) + 2$ in some run r. Therefore, p_j does not receive a COMMIT message from a majority of processes (including from itself and the leader) in round $GSR(r) + 2$. Since, in our model, from $GSR(r)$ onward, every correct process receives messages from a majority of correct processes (including itself and the leader), it must have received at least one message with type t other than COMMIT. t cannot be DECIDE, since p_j didn't decide in round $GSR(r) + 2$. Therefore, t must be PREPARE. Therefore, there must be a process p_i that sent in round $GSR(r) + 2$, a message with $msgType =$ PREPARE.

In round $GSR(r) + 1$, p_i does not decide or commit, since it sent a PREPARE message in the next round. Therefore, one of the commit rules evaluates to $false$ for p_i. It is not *commit-1* or *commit-3*, because all correct processes agree on the identity of the leader from $GSR(r)$ onward, and each process receives a message from a majority of processes (*commit-1*), and starting from round $GSR(r) + 1$, the rule *commit-3* evaluates to $true$ as was explained in the description of the algorithm.

Therefore, *commit-2* must be the rule that evaluates to $false$. The only possible reason for this is that the leader indicated $lastApproval \neq GSR(r)$ in its round $GSR(r) + 1$ message. If $GSR(r) = 0$ we get a contradiction since $lastApproval$ is initialized to 0. Otherwise ($GSR(r) > 0$), notice that the leader couldn't have decided by start of round $GSR(r)$, since otherwise all correct processes would decide by end of $GSR(r)$. Therefore, according to our algorithm, the leader had to set $lastApproval = GSR(r)$ in round $GSR(r)$ (since every process hears from a majority starting at round $GSR(r)$), and this is a contradiction. □

Chapter 7

Linear Lower Bound for ◊SR

We use the notion of k-round reducibility, to prove that at least n rounds starting at GSR are needed to solve consensus in the $\Diamond SR$ model. We formally define the $\Diamond SR$ model as follows:

$\Diamond SR$ *(Strong-Reliable)*: $t < n/2$, reliable links, $\Diamond S$ failure detector, the unsuspected process is $\Diamond n$-source and all correct processes are $(n-f-1)$-destinations$_v$, where $f < \frac{n}{2} - 1$).

Lemma 7.1. *Any model $M_{\Diamond S}$ that requires a $\Diamond S$ failure detector and environment properties $\mathcal{P}$ is 0-round reducible to a model $M_{\Diamond n}$ that assumes a correct $\Diamond n$-source process and $\mathcal{P}$, i.e. $M_{\Diamond n} \geq_0 M_{\Diamond S}$.*

Proof. We implement the reduction algorithm $T_{M_{\Diamond n} \rightarrow M_{\Diamond S}}$ as follows: *compute$_T$()* receives a multi-set of messages M received so far, and the current round number k, but no oracle output (since $M_{\Diamond n}$ does not include an oracle) and produces the set of suspected processes FD_T as follows: $FD_T \leftarrow \{ \ j \mid M[k][j] = \perp \ \}$. It then passes M, k and FD_T to *compute$_A$()*. *initialize$_T$()* calls *initialize$_A$()* with $\emptyset$ as the set of suspected processes. Since in every round $k' \geq GSR_{M_{\Diamond n}}$ there exists one process (the $\Diamond n$-source correct process) whose k' round message reaches every correct process by the end of round k', this process is not included in any of the FD_T sets produced by algorithm $T_{M_{\Diamond n} \rightarrow M_{\Diamond S}}$ at any process in round k', i.e. is not suspected. Since no faulty process enters round $GSR_{M_{\Diamond n}}$, no such process sends a round k' message, and thus every faulty process is suspected from round $GSR_{M_{\Diamond n}}$ onward. Therefore, the produced set FD_T satisfies the specification of $\Diamond S$ in our framework such that the eventual properties of $\Diamond S$ are satisfied from $GSR_{M_{\Diamond n}}$ onward. Since M (the message set) and k are not altered by $T_{M_{\Diamond n} \rightarrow M_{\Diamond S}}$, all the other properties $\mathcal{P}$ are still preserved from round $GSR_{M_{\Diamond n}}$ onward. Therefore, $GSR_{M_{\Diamond n}} = GSR_{M_{\Diamond S}}$ and $M_{\Diamond n} \geq_0 M_{\Diamond S}$. □

From Lemma 7.1, it follows that it suffices to prove the lower bound for a model just like $\Diamond SR$, but without the assumption of $\Diamond S$. We denote this model by $\Diamond SR \backslash \Diamond S$.

We prove the lower bound using the impossibility of consensus in the *mobile failure* model [46], in which no process crashes, and in each communication step there is one process whose messages may be lost.

Below we denote the prefix of length l rounds of a run r by $r(l)$.

Lemma 7.2. *For any $k \in N$, let r be a run in the mobile failure model. There exists a run r' in $\Diamond SR \backslash \Diamond S$ with GSR(r') $= k$ and $f = 0$ such that $r'(k + n - 2) = r(k + n - 2)$.*

Proof. We construct r' as follows: (i) $f = 0$ and $GSR(r') = k$, (ii) r' is identical to r in the first $k + n - 2$ rounds, except that messages are delayed to round $k + n - 1$ instead of being lost, and (iii) from round $k + n - 1$ onward, r' is synchronous (all links are timely).

We show that r' is a run in model $\Diamond SR \backslash \Diamond S$. In each round of $r'(k+n-2)$, a subset of messages sent by at most one process is delayed and all other messages arrive in the same round in which they are sent, and from round $k + n - 1$ onward, no message is delayed in r'. Therefore, in r', each process receives messages from at least $n - 1$ processes in every round and is therefore an $(n - f - 1)$-destination$_v$ (recall that $f = 0$ in r'). Since $r'(k + n - 2)$ lasts only $n - 1$ rounds starting from $GSR(r')$ (and there are n processes), there exists some correct process whose messages are not delayed in any round from $GSR(r')$. This process is a correct $\Diamond n$-source in r'. Finally, since every message sent before round $k + n - 1$ in r' arrives at the latest in round $k + n - 1$ and every message sent in later round arrives in the same round in which it is sent, we conclude that links are reliable in r'. □

We strengthen the lower bound by proving that it is impossible to reach global decision in less than n rounds from GSR in the $\Diamond SR \backslash \Diamond S$ model, even with an algorithm especially tailored for some specific GSR.

Lemma 7.3. *For $k \in N, k \geq 1$, no algorithm exists that in every run r in which GSR(r) $= k$ achieves global decision before round GSR(r)+$(n-1)$, in the $\Diamond SR \backslash \Diamond S$ model.*

Proof. For $k \in N, k \geq 1$, assume there exists an algorithm A_k that solves consensus in $\Diamond SR \backslash \Diamond S$, and in every run with GSR$=$ k reaches global decision by round $k+n-2$. Then we run A_k in the mobile model for $k+n-2$ rounds. Denote this run by r. From Lemma 7.2, there is a run r' in $\Diamond SR \backslash \Diamond S$ with $GSR(r') = k$ and $f = 0$, such that $r'(k+n-2) = r(k+n-2)$. Therefore, A_k cannot distinguish r from r' in the first $k+n-2$ rounds and decides by round $k+n-2$ in r as it does in r'. We conclude that A_k reaches a global decision for every run r in the mobile failure model. A contradiction to [46]. □

Note that our proof (combined with Algorithm 2, which achieves global decision by GSR+2 in $\Diamond LM$) immediately implies that $\Diamond SR \not\succeq_k \Diamond LM$ for any $k < n-3$, since otherwise, we could use the reduction algorithm to simulate $\Diamond LM$ in $\Diamond SR$ in any run r with $GSR_{\Diamond LM} < GSR_{\Diamond SR}(r) + n - 3$ and use the algorithm in Figure 6.1 on top of the reduction algorithm. Since the algorithm in Figure 6.1 assures global decision by $GSR_{\Diamond LM}(r) + 2$ we get that there exists an algorithm that for any run r achieves global decision before round $GSR_{\Diamond SR}(r) + n - 1$, a contradiction to our lower bound.

Chapter 8

Constant-Time Algorithm in $\Diamond AFM$

In this chapter, we investigate whether constant time decision is possible without an oracle in a model weaker than ES. We are not aware of any previous constant-time algorithms for such a model.

In the $\Diamond$AFM model, each process has timely incoming links from a correct majority of processes, and a majority of timely outgoing links (from GSR onward), both can vary in each round. The number of outgoing links may decrease if more incoming links are timely. Formally:

$\Diamond AFM$ *(All-From-Majority)*: $t < n/2$, $\exists m \in N$, $f \leq m < n/2$ such that every correct process is a $\Diamond(n-m)$-destination$_v$ and a $\Diamond(m+1)$-source$_v$. Note that m can be different in each run.

8.1 Algorithm

Figure 8.1 presents a majority-based algorithm for $\Diamond$AFM, which always reaches global decision by round GSR+5. In runs with GSR $= 0$, this means that consensus is achieved in 5 rounds. In runs with GSR > 0, global decision is reached in 6 rounds. At the end of this chapter we present an optimization of the algorithm for the case of $n = 2m + 1$ (i.e., when both $(m + 1)$ and $(n - m)$ are majorities), and in the following sections we prove that the optimized algorithm reaches global decision by round GSR+4 for $n = 2m + 1$ (when GSR > 0), and by round GSR+5 for other values of m ($f \leq m < n/2$). The

```
1: Additional state
2:     $est_i$, $maxEST_i \in Values$, initially $prop_i$
3:     $ts_i$, $maxTS_i \in \mathbf{N}$, initially 0
4:     $IgotCommit_i \in Boolean$, initially $false$
5:     $gotCommit_i \in 2^{\Pi}$, initially $\emptyset$
6:     $msgType_i \in \{$PREPARE, PRE-COMMIT, COMMIT, DECIDE$\}$, initially PREPARE
7: procedure initialize()
8:     return message $\langle msgType_i, est_i, ts_i, IgotCommit_i, gotCommit_i \rangle$    /*round 1 message*/
9: procedure compute($k_i$, M[*][*])
10:    if $dec_i = \bot$ then
11:            /*Update variables*/
12:        $maxTS_i \leftarrow \mathbf{max}\{ m.ts \mid m \in M[k_i][*] \}$
13:        $maxEST_i \leftarrow \mathbf{max}\{ m.est \mid m \in M[k_i][*] \wedge m.ts = maxTS_i \}$
14:        $IgotCommit_i \leftarrow \exists m \in M[k_i][*]$ s.t. $m.msgType =$ COMMIT
15:        $gotCommit_i \leftarrow \{ j \mid M[k_i][j].IgotCommit \}$
16:            /*Round Actions*/
17:        if $\exists m \in M[k_i][*]$ s.t. $m.msgType =$ DECIDE then    /*decide-1*/
18:            $dec_i \leftarrow est_i \leftarrow m.est$; $msgType_i \leftarrow$ DECIDE
19:        else if $|\{ j \mid M[k_i][j].msgType =$ COMMIT $\}| > \lfloor n/2 \rfloor \wedge M[k_i][i].msgType =$ COMMIT
           then  /*decide-2*/
20:            $dec_i \leftarrow est_i$; $msgType_i \leftarrow$ DECIDE
21:        else if $|\bigcup_{j \in \Pi} M[k_i][j].gotCommit| > \lfloor n/2 \rfloor$ then    /*decide-3*/
22:            $dec_i \leftarrow est_i \leftarrow maxEST_i$; $msgType_i \leftarrow$ DECIDE
23:        else if $|\{ j \mid M[k_i][j].est = maxEST_i \}| > \lfloor n/2 \rfloor$ then    /*pre-commit*/
24:            if $\exists j$ s.t. $M[k_i][j].est = maxEST_i \wedge M[k_i][j].msgType =$ COMMIT or PRE-COMMIT then
               /*commit*/
25:                $est_i \leftarrow maxEST_i$; $ts_i \leftarrow k_i$; $msgType_i \leftarrow$ COMMIT;
26:            else
27:                $est_i \leftarrow maxEST_i$; $ts_i \leftarrow maxTS_i$; $msgType_i \leftarrow$ PRE-COMMIT;
28:        else
29:            $ts_i \leftarrow maxTS_i$; $est_i \leftarrow maxEST_i$; $msgType_i \leftarrow$ PREPARE
30:    return message $\langle msgType_i, est_i, ts_i, IgotCommit_i, gotCommit_i \rangle$    /*round $k_i + 1$ message*/
```

Figure 8.1: Majority–based algorithm for $\Diamond AFM$ model. Code for process p_i. Optimization for $n = 2m + 1$ is marked in gray.

code used for optimization is marked in gray in Figure 8.1 and should be ignored until its explanation in the next section.

In general, the algorithm in Figure 8.1 is similar to the leader-based algorithm presented in Chapter 6. We therefore focus mainly on the differences from the leader-based algorithm. Since $\Diamond AFM$ does not assume a failure detector, the oracle's output is not a parameter for *compute()*.

The variables maintained by each process p_i are similar to those of the algorithm in Chapter 6. A new variable, $maxEST_i$, holds the maximal estimate received with timestamp $maxTS_i$ in the current round (recall that *Values* is a totally ordered set). A new message type is introduced, PRE-COMMIT. Intuitively, pre-committing is similar to committing, but without increasing the timestamp. An estimate must be pre-committed by some process before it is committed.

Pre-commit is needed, since, unlike $\Diamond$LM, where the leader is a $\Diamond n$-source, $\Diamond$AFM never assures that a process is able to convey information to all other processes in a single round. If we hadn't introduced PRE-COMMIT, it would have been possible for two different estimates to be committed in alternating rounds, where a majority of processes hear and adopt estimate est_1, (which has the maximal timestamp) but some other process does not hear est_1 and commits to est_2, increasing its timestamp. In the next round the situation flips, and est_2 is adopted by a majority while est_1 is committed, and so on, precluding decision.

In $\Diamond$AFM, in every round from GSR onward, each process hears from $(n - m)$ correct processes, and its outgoing message reaches $m + 1$ processes. Note that the $m + 1$ processes the message reaches overlaps the set of $(n - m)$ correct processes every other process hears from in the next round, allowing information to propagate to all correct processes in two rounds. Thus, a single *pre-commit* phase suffices to eliminate races as described above, where two different values are repeatedly committed after GSR.

We now describe p_i's computation. If p_i does not decide, it evaluates the following two conditions: *pre-commit* (line 23): p_i receives messages from a majority of processes with $maxEST_i$ as their estimate; and *commit* (line 24): at least one COMMIT or PRE-COMMIT message is received with $maxEST_i$. If both conditions are true, then p_i sets its message type (for the round $k_i + 1$ message) to COMMIT, adopts the estimate $maxEST_i$, and sets its timestamp to the current round number k_i (line 25). We say that p_i ***commits in round*** k_i with estimate $maxEST_i$. If, however, only the first condition holds, then p_i sets its message type to PRE-COMMIT, adopts the estimate $maxEST_i$, and sets its timestamp to $maxTS_i$ (line 27). We say that p_i ***pre-commits in round*** k_i with estimate $maxEST_i$. If neither condition holds, p_i prepares (sets his message type to PREPARE) and adopts the estimate $maxEST_i$ and timestamp $maxTS_i$ (line 29).

8.2 Optimization for $n = 2m + 1$

We present an optimization of the algorithm for the case of $n = 2m + 1$ (i.e., when both $(m + 1)$ and $(n - m)$ are majorities). The additional code used for the optimization is marked in gray in Figure 8.1. In the following sections, we prove that the optimized algorithm reaches global decision by round GSR+4 (five rounds) for $n = 2m + 1$ (when GSR > 0), and by round GSR+5 (six rounds when GSR > 0 and five when GSR $= 0$) for other values of m ($f \leq m < n/2$).

The optimization relies on the *IgotCommit* and *gotCommit* variables, that are used for "gossiping" about COMMIT messages. Whenever a process receives a COMMIT message, it indicates this in its next round message by setting *IgotCommit* to *true*. In order to have all processes learn about commits, we use the *gotCommit* message field. A process includes in the *gotCommit* set that it sends in round $k + 1$, all processes that it knows have gotten COMMIT messages in round $k - 1$ (based on *IgotCommit* indications sent in round k). Thus, in round $k + 1$, the incoming *gotCommit* sets from different processes can give p_i a better picture about which processes got COMMIT messages in round $k - 1$. In the following sections of this chapter, we prove that if the union of the *gotCommit* groups that a process gets exceeds $\lfloor n/2 \rfloor$, it is safe for the process to decide on $maxEST$ (rule decide-3) and this optimization allows us to speed up global decision to be by round GSR+4 instead of by round GSR+5, when GSR > 0 (i.e., to decide in 5 instead of 6 rounds). Note that if $GSR = 0$ or $GSR = 1$ the unoptimized algorithm takes 5 rounds. We formally prove the correctness of the optimized algorithm in Figure 8.1 in the following section.

Theorem 8.1. *The algorithm solves consensus in our model with global decision by round* $GSR(r) + 5$ *(or* $GSR(r) + 4$ *in case* $n = 2m + 1$*).*

Proof. From Lemma 8.14, every correct process decides by round $GSR(r) + 4$, if $n = 2m + 1$. From Lemma 8.15, every process decides by round $GSR(r) + 5$. Validity holds, since the decision can only be one of the initial estimates of the processes. Uniform agreement is proven in Lemma 8.8. $\square$

8.3 Correctness

We first give informal explanation of correctness. A process may commit with different estimates in different rounds. However, we show that starting from a round k in which a majority of processes M commit with some estimate x onward, every commit is with estimate x. Note that this implies agreement, since decision is impossible before a majority of processes commit (see decision rules). To understand why this is true, note first that by rule *pre-commit*, all COMMIT and PRE-COMMIT messages sent in the same round are with the same estimate. This explains why a commitment with $y \neq x$ is impossible in round k. Additionally, note that a process's timestamp never decreases, and therefore the processes in M have timestamps $\geq k$ in subsequent rounds. Suppose that a process p_i commits in round $k' > k$. Rule *pre-commit* ensures that p_i hears from a majority. Since every two majorities intersect, p_i hears from at least one process in M. Since p_i commits on $maxEST_i$, which has the maximal timestamp, p_i commits with a timestamp $\geq k$. Using an inductive argument, we get that $maxEST_i = x$. Since no decision is made before a majority commits, and every decision is either on the value of a previous decision (rule *decide-1*), or on the value sent in COMMIT messages (rule *decide-2*), which equals x from round k onward, all decisions are with x.

We now formally prove correctness.

Lemma 8.2. *A process's timestamp at the start of round k is less than k.*

Proof. We prove the claim by induction on the round number k'. Base case: $k' = 1$. The claim is correct since a process's timestamp is initialized to 0. The induction hypothesis is that the claim holds up to round k'. Let us inspect the possible actions of a process at the end of round k'. A process can decide and in this case its timestamp does not change and in round $k' + 1$ it will remain less or equal to $k' - 1$, by the induction hypothesis. Alternatively, a process may commit, and then (on line 25) it will adopt k' as its new timestamp for round $k' + 1$, and the claim holds here as well. Finally, a process may adopt the timestamp of a round k' message it received in round k' (on line 27 or 29) and again, by induction hypothesis, the claim is $true$. □

Lemma 8.3. *A process's timestamp is non-decreasing.*

Proof. Observe that when a process decides, its timestamp does not change. It does not change in the following rounds as well. If a process p_i does not decide in round k, then it can change its timestamp by adopting either k (when committing on line 25) or the maximum timestamp (of a round k message) received in round k as its new timestamp (on line 27 or 29). Since p_i receives its own message in round k, the latter is not lower than its current timestamp. In case it commits, since according to Lemma 8.2, its old timestamp cannot exceed $k - 1$, by adopting k it can only increase. □

Lemma 8.4. *For every round k, no two processes commit or pre-commit with different estimates in round k.*

Proof. Consider two processes p_i and p_j that commit or pre-commit in round k with estimates est_i and est_j. Thus, by *pre-commit* rule, each of them has received in round k a majority of messages that contain est_i and est_j, respectively. As two majorities intersect, $est_i = est_j$. Therefore, p_i and p_j commit or pre-commit with the same estimate. □

Lemma 8.5. *If some process sends a message other than* DECIDE *with timestamp $ts > 0$ and estimate x, then some process commits in round ts with estimate x.*

Proof. We prove the claim by induction on the round number k', starting from a round k_0 in which a message other than DECIDE with the timestamp ts is first sent with some estimate x' by some process p_j.

Base Case. $k' = k_0$. From the definition of k_0, p_j could not receive a message with ts from another process in an earlier round. Thus, p_j commits with timestamp ts and estimate x' in round $k_0 - 1$, and from the algorithm, $k_0 - 1 = ts$.

Induction Hypothesis. If any process sends a PREPARE, PRE-COMMIT or COMMIT message in round $k1$, such that $k_0 \leq k1 \leq k'$, with timestamp ts and some estimate x'', then some process commits in round ts with estimate x''.

Induction Step. We need to show that if, in round $k' + 1$, a process sends a message other than DECIDE with timestamp ts and some estimate x'' then some process commits in round ts with estimate x''. Observe, that if a COMMIT message is sent, it has a timestamp equal to the previous round number k', and since $ts = k_0 - 1 < k'$ (from the base case), this case is not possible. Observe that if a PREPARE or PRE-COMMIT message is sent in round $k' + 1$ with timestamp ts and estimate x'', the sending process must have adopted the timestamp together with the estimate from some PREPARE, PRE-COMMIT, or COMMIT message sent in round k' (this message couldn't have been DECIDE since otherwise the $k'+1$ round message would be DECIDE and not PREPARE). By the induction hypothesis, we get that some process commits in round ts and estimate x''. □

Please note that the claim in Lemma 8.5 does not hold for DECIDE messages, since a process can decide adopting only the estimate and not the associated timestamp from another DECIDE message.

Lemma 8.6. *If rule* decide-3 *evaluates to true in some round k, there exists a majority of processes that receive a* COMMIT *message in round* $k - 2$.

Proof. Suppose rule *decide-3* evaluates to true in some round k at process p_i. Therefore, the union of the gotCommit sets p_i receives in round k messages includes more than $\lfloor (n/2) \rfloor$ indices. These $gotCommit$ groups were created in round $k - 1$ by the processes that sent these messages, according to the $IgotCommit$ values that these processes received. The fact that the union of the $gotCommit$ groups has size $> \lfloor (n/2) \rfloor$ indicates that more than $\lfloor (n/2) \rfloor$ messages were received in round $k - 1$ with $IgotCommit = true$ from different processes. A process sends a message with $IgotCommit = true$ only when it receives a COMMIT message in the previous round. Therefore, more than $\lfloor (n/2) \rfloor$ (a majority) of processes received a COMMIT message in round $k - 2$. □

Lemma 8.7. *(a) If some process receives a* COMMIT *message in round k with estimate x, and some process commits in round k with estimate z, then $z = x$ (b) if a process p_i commits to x in round k, or receives a* COMMIT *message with estimate x in round k, and does not decide in this round, then it adopts x as its estimate with timestamp $ts \geq k - 1$.*

Proof. (a) If some process commits with estimate z in round k, it must have received a COMMIT or PRE-COMMIT message with z (rule *commit*), and according to Lemma 8.4, all such messages have the same estimate, and therefore $z = x$. (b) If p_i commits x, then it sets its timestamp to k and adopts x as its estimate. If p_i receives a COMMIT message with estimate x, it cannot commit or pre-commit on a different value since according to rule *pre-commit* a process can commit or pre-commit only on a value received with the highest timestamp. Moreover, p_i receives x with the timestamp $k-1$ (which is maximal at round k) and (Lemma 8.4) every message with this timestamp has x as estimate. Since it does not commit on x either, it does not commit at all in round k. Since p_i does not decide in this round, it must either pre-commit or prepare with the estimate x and adopt its timestamp: $k-1$. □

Lemma 8.8 (Uniform Agreement)**.** *Let k be the first round in which there exists a group consisting of a majority of processes such that each process of the group either commits or receives a* COMMIT *message. Then, no decision is made before round $k+1$, and all decisions and commitments made in rounds $k' \geq k-1$ are with the same estimate.*

Proof. Let k be the lowest numbered round in which each one out of a majority of processes either commits x (from Lemma 8.4 all commitments in some round are with the same value) or receives a COMMIT message with a value x (x is well defined according to Lemma 8.7). Denote this group of processes by S_x. According to Lemma 8.7, every process in S_x has timestamp $\geq k-1$ at the end of round k (we prove below that a decision is not possible in round k). Note also, that $k-1>0$. This is true since by definition of k, processes in S_x either commit or receive a COMMIT message in round k. Therefore, in round $k-1$ some process must either commit or pre-commit and since round numbering starts from 1, we have that $k-1>0$.

There are three decision rules in the algorithm. We show that none of them could evaluate to true for any process before round $k+1$. Let k' be the first round in which any process decides. Rule *decide-1* may be true only after some process has already decided, and thus cannot cause the first decision. Rule *decide-2* can evaluate to true only if a majority of processes committed in the previous round. Since the first round in which this happens is k, this rule may evaluate to true only starting from round $k+1$. The last one is rule *decide-3*. According to Lemma 8.6, if this rule evaluates to true in round k', there must have been a majority of processes that received a COMMIT message in round $k'-2$. Since the first round in which this could happen is k, we get that $k' \geq k+2$. So in any case, no decision is possible before round $k+1$. We now prove that all decisions and commitments made in rounds $k' \geq k-1$ are with estimate x.

Base Case. $k' = k - 1$. As proven above, no process decides in round $k' < k + 1$. Assume by contradiction that some process commits on a value $z \neq x$ in round $k - 1$. By Lemma 8.4, no process can commit or pre-commit on x in the same round. Therefore, in round k, no process receives a COMMIT or PRE-COMMIT message with the estimate x. Thus, no process commits x in round k (rule *commit*). This contradicts the definition of k.

Induction Hypothesis. If any process commits or decides in any round $k1$ such that $k-1 \leq k1 \leq k'$, then it commits with estimate x or decides x.

Induction Step. decision in round $k' + 1$. Suppose some process p decides in round $k' + 1$. If it decides using rule *decide-1* or *decide-2*, then in that round either some other process sends a DECIDE message with decision value y or p sends a COMMIT message with estimate y. In both cases, by the induction hypothesis, $y = x$.

If it decides by rule *decide-3*, then according to Lemma 8.6, there must be a majority of processes that receive COMMIT messages two rounds earlier, in round $k' - 1$. Since the first round in which this can happen is k, we have that $k' - 1 \geq k$. According to the induction hypothesis, the commit messages received are with estimate x. Therefore, in round $k' - 1$, some majority of processes M received a COMMIT message with the estimate x. According to Lemma 8.7, if a process in M does not decide in round $k' - 1$, it will adopt x with timestamp $\geq k' - 2$. By the induction hypothesis, every process that decides in round $k' - 1$ or k', decides x and no process commits with a different value in round $k' - 1$ or $k' - 2$. Therefore, in round k', all estimates different from x are sent with a timestamp $< k' - 2$. No process can commit or pre-commit on an estimate other than x in round k' since x is the value processes in M send and every two majorities intersect (rule *pre-commit* must be false for any other value).

p receives a round $k' + 1$ message from at least one process p_i that receives a round k' message from some process in M. Therefore, p_i receives a round k' message with the estimate x and timestamp $\geq k' - 2$. As was explained above, no other estimate can have a timestamp that high in round k', so if p_i prepares or pre-commits, it must be with estimate x. If p_i commits, it is with the estimate x as well, according to the induction hypothesis. p_i does not decide, since otherwise p would decide by rule *decide-1* and not *decide-3*. Therefore, p_i sends a round $k'+1$ message with the estimate x and a timestamp $\geq k'-2$. Since no process can commit on a value different than x in round k' or $k'-1$, this timestamp is higher than the timestamp of any other estimate sent in round $k'+1$. Therefore $maxEST$ of p must be equal to x. Therefore, p decides x.

commit in round $k'+1$. Suppose by contradiction that some process p_j commits in round $k'+1$ with estimate $z \neq x$. Then p_j does not receive any DECIDE message in round $k'+1$. Also note that according to rule *pre-commit*, p_j commits on an estimate that it receives with the highest timestamp: $maxTS$. Therefore, some process sends a round $k'+1$ message with timestamp $maxTS$ and estimate z. By Lemma 8.2, the highest timestamp that can be received in round $k'+1$ is k', and therefore $maxTS \leq k'$. Since p_j commits in round $k'+1$, it receives round $k'+1$ messages from a majority of process (rule *pre-commit*) and hence, receives a round $k'+1$ message from at least one process $p_i \in S_x$. According to Lemma 8.7, p_i has at least timestamp $k-1$ at the end of round k. By Lemma 8.3, p_i's timestamp is at least $k-1$ and therefore $maxTS \geq k-1$. Thus, we have $k-1 \leq maxTS \leq k'$. Since $k-1 > 0$ (as shown above), and since p_j does not receive any DECIDE messages in round $k'+1$, by Lemma 8.5 there is a process that commits z in round $maxTS$. By the induction hypothesis, every process that commits in round $maxTS$ commits $x \neq z$; a contradiction. □

8.4 Performance

We start by informally explaining why the algorithm decides by round GSR+5. First, if some process decides by round GSR+3, then its DECIDE message reaches every process by the end of round GSR+5. Assume no process decides by GSR+3. Second, if no process commits in round GSR, the maximum timestamp sent in GSR is the same as the maximum timestamp sent in round GSR+1, and it reaches every correct process by the end of round k_1 =GSR+1, at which point all processes have the same $maxEST$. Finally, if a process commits in GSR (note that this cannot happen if GSR $= 0$ or GSR $= 1$), the use of pre-commit ensures that no different value is committed in GSR+1, and thus this value has the highest timestamp among those sent in round GSR+2, and this timestamp and its estimate reach every process by the end of round k_2 =GSR+2. In both cases, every process has the same $maxEST$ at the end of round $k = k_1$ or $k = k_2$. Thus, all processes send the same estimate in round $k + 1$, and in the ensuing round, a majority of processes receives it and pre-commits (at least). In round $k + 2$, every correct process receives the same estimate from majority and a PRE-COMMIT or COMMIT message, and commits. Finally, by round $k+3$, which is at most GSR+5, every process decides by rule *decide-2*.

We now give a formal proof.

Lemma 8.9. *If* $n = 2m + 1$*, in every run* r*, if some process* p *commits in round* $GSR(r)$ *with an estimate* x*, then all processes decide by the end of round* $GSR(r) + 3$*.*

Proof. Suppose that some process p_i does not decide by the end of round $GSR(r) + 3$. This means that it evaluates rules *decide-1*, *decide-2* and *decide-3* to $false$. Therefore, p_i does not receive any DECIDE message, and $|\bigcup_{j\in\Pi} M[k_i][j].gotCommit| \leq \lfloor n/2 \rfloor$. p_i receives a round $GSR(r) + 3$ message from a group M of $(n - m)$ processes. Thus, there are $(n - m)$ processes that together receive in round $GSR(r) + 2$ messages with $IgotCommit = true$ from at most $\lfloor n/2 \rfloor$ processes. Every process in M does not receive a DECIDE message in round $GSR(r)+2$, since otherwise their next round message would be DECIDE. Since each process's message reaches $(m + 1)$ from $GSR(r)$ onward, it reaches at least one process from any group of $(n - m)$ processes. Therefore, the number of processes that send a message with $IgotCommit = true$ in round $GSR(r) + 2$ is at most $\lfloor (n/2) \rfloor$. We get that in round $GSR(r) + 1$, at most $\lfloor (n/2) \rfloor$ processes received a COMMIT message. Every process whose message reaches a process in M does not decide in round $GSR(r) + 1$, since otherwise their next round message would be DECIDE and processes in M do not receive any such messages. Since p sends a COMMIT message, its message should reaches at least $m + 1$ processes (a majority when $n = 2m + 1$). As explained above, we get a contradiction the assumption that p sends a COMMIT message in round $GSR(r) + 1$. □

Notations: (relating to a specific run r)

$absMaxTS(k)$ = max{ $m.ts$ | message m is sent in round k}

$absMaxEST(k)$ = max{ $m.est$ | message m is sent in round k s.t. $m.ts = absMaxTS(k)$}

Lemma 8.10. *In every run r, if no correct process decides by the end of round $k \geq GSR(r)$, then in round k at least $m+1$ correct processes adopt the estimate $absMaxEST(k)$ with timestamps equal to $absMaxTS(k)$ or to k (in case it is adopted by committing).*

Proof. Let us observe round k messages.

Denote by p_{max} the (correct) process that sends $absMaxEST(k)$ with the timestamp $absMaxTS(k)$. By the assumptions of our model, the round k message $\langle *, absMaxEST(k), absMaxTS(k), *, * \rangle$ will reach at least $m+1$ correct processes. Denote the group of processes that actually get this message by A. Note that since a process receives a subset of all messages sent in round k, for any $p_i \in A$, $maxTS_i = absMaxTS(k)$ and $maxEST_i = absMaxEST(k)$. The conditions of the lemma assume that no correct process decides by the end of round k. Therefore, each process $p_i \in A$ must commit, pre-commit or just prepare for the next round. If p_i commits or pre-commits, rule *pre-commit* must hold for it. This rule makes sure that the estimate p_i adopts is equal to $maxEST_i$. Therefore, p_i will adopt $absMaxEST(k)$. If p_i prepares, it will execute line 29 of the pseudo-code, adopting $absMaxEST(k)$ as well. Therefore all the processes in A (at least $m+1$ processes) will adopt the same estimate $absMaxEST(k)$. Observe, that they will either adopt it with timestamp $absMaxTS(k)$ (if they pre-commit or prepare) or with timestamp k if they commit. □

Lemma 8.11. *In every run r, if no correct process decides by the end of round $GSR(r)+1$, and no process commits in round $GSR(r)$, all processes will have the same estimate by the end of round $GSR(r)+1$.*

Proof. By Lemma 8.10, at the end of round $GSR(r)$, at least $m+1$ of processes adopt the estimate $absMaxEST(GSR(r))$ with a timestamp equal to $absMaxTS(GSR(r))$. Notice that an estimate $est' \neq absMaxEST(GSR(r))$ can become $absMaxEST(GSR(r)+1)$ only by adopting a new timestamp (that was not sent in round $GSR(r)$). This can be done only if a process commits with est' in round $GSR(r)$, and this is not possible by the assumptions of our lemma. We conclude that $absMaxEST(GSR(r)+1) = absMaxEST(GSR(r))$ ($\neq est'$).

No process decides in round $GSR(r)+1$, and each p_i process receives a round $GSR(r)+1$ message from $n-m$ processes, including one message of the form $\langle *,$ $absMaxEST(GSR(r)+1)$, ts, $*$, $*\rangle$ and ts is either equal to $absMaxTS(GSR(r))$ ($ts \neq GSR(r)$ since we assume in this lemma that no process commits in round $GSR(r)$). Whether p_i commits, pre-commits or prepares, because of rule *pre-commit* and line 29, the estimate p_i adopts is equal to $maxEST_i$. Therefore, p_i will adopt $absMaxEST(GSR(r)+1)$, and we get that all processes adopt the same estimate by the end of round $GSR(r)+1$. □

Lemma 8.12. *In every run r, if no correct process decides by the end of round $GSR(r)+2$, and some process commits in round $GSR(r)$, all processes will have the same estimate by the end of round $GSR(r)+2$.*

Proof. By Lemma 8.10, at the end of round $GSR(r)+1$, at least $m+1$ of processes adopt the
estimate $absMaxEST(GSR(r)+1)$ with a timestamp equal to $GSR(r)$. Notice that an estimate $est' \neq$
$absMaxEST(GSR(r)+1)$ can become $absMaxEST(GSR(r)+2)$ only by adopting a new timestamp (that was not sent in round $GSR(r)+1$). This can be done only if a process commits with est' in round $GSR(r)+1$, and this is not possible because some process will receive the COMMIT message sent in this round, and Lemma 8.7. We conclude that $absMaxEST(GSR(r)+2) = absMaxEST(GSR(r)+1)$ ($\neq est'$).

No process decides in round $GSR(r)+2$, and each p_i process receives a round $GSR(r)+2$ message from $n-m$ processes, including one message of the form $\langle *,$ $absMaxEST(GSR(r)+2)$, ts, $*$, $*\rangle$ and ts is either equal to $GSR(r)$ or to $GSR(r)+1$. Whether p_i commits, pre-commits or prepares, because of rule *pre-commit* and line 29, the estimate p_i adopts is equal to $maxEST_i$. Therefore, p_i will adopt $absMaxEST(GSR(r)+1)$, and we get that all processes adopt the same estimate by the end of round $GSR(r)+2$. □

Lemma 8.13. *If in a round $k \geq GSR(r)$ all estimates being sent are the same, all correct processes decide by round $k+2$.*

Proof. Observe that in our model every correct process executes an infinite number of rounds, and in particular, executes round $k+2$. Also, it is obvious that all estimates being sent remain the same in all rounds starting at k. We prove the lemma by contradiction. Assume that some correct process p_j does not decide by round $k+2$ in some run r. Therefore, p_j couldn't have received a COMMIT message from a majority of processes in round $k+2$. Since, in our model, from $GSR(r)$ onward, every correct process receives messages from a majority of correct processes (including itself), it must have received a round $k+2$ message m s.t. $m.msgType = t$ from some process p_i with type $t \neq COMMIT$. $t \neq$DECIDE, since p_j didn't decide in round $k+2$. Therefore, t must be PREPARE or PRE-COMMIT. If $t =$PREPARE, this can happen only if in round $k+1$, process p_i received messages with different estimates, since otherwise (if all estimates it receives are the same), even if there were no proper conditions (according to the algorithm) for p_i to DECIDE or COMMIT, its PRE-COMMIT rule would definitely evaluate to true and its round $k+2$ message would be (at least) PRE-COMMIT. Therefore, $t =$PREPARE is a contradiction to our assumption that in round $k+1$ all estimates being sent are the same. If $t =$PRE-COMMIT, this means that p_i didn't receive any DECIDE message in round $k+1$, and that rule *pre-commit* evaluated to $true$ for p_i, but rule *commit* did not. Therefore, p_i received $k+1$ round messages from a majority of processes with some estimate $maxEST_i$, but didn't receive any of them with the type COMMIT or PRE-COMMIT. This means that at the end of round k, there were processes that didn't PRE-COMMIT. Lets observe one such process p_c (who's round $k+1$ message reached p_i) at the end of round k. It couldn't have decided since p_i didn't receive any DECIDE messages in round $k+1$. Since all estimates sent are the same in round k, its rule *pre-commit* must evaluate to true at the end of round k, and it sends either a COMMIT or a PRE-COMMIT message in round $k+1$, a contradiction to the fact that p_i received no such messages (since starting with round k, all estimates are the same, est' must be the estimate sent by p_c). □

Lemma 8.14. *If $n = 2m+1$ then in every run r in which GSR > 0, all correct processes decide by round $GSR(r)+4$, i.e., in 5 rounds.*

Proof. If some process correct process decides by the end of round $GSR(r)+1$, lets denote by k the round in which this happens, or $GSR(r)-1$ (the later round between the two). Since $k \geq GSR(r)-1$, in round $k+1$, it is assured that the decision message will reach $m+1$ processes, and in round $k+2$, it will reach all the process since each one receives a message from $n-m$ processes. Therefore, every process will decide by round $k+2$. If $k = GSR(r)+1$, $k+2 = GSR(r)+3$, and the lemma holds.

Suppose that no correct process decides by the end of round $GSR(r)+1$. If some process commits in round $GSR(r)$, all processes will decide by round $GSR(r)+3$, by Lemma 8.9. If no process commits in $GSR(r)$, by Lemma 8.11, all processes will adopt the same estimate by the end of round $GSR(r)+1$, and send it in round $GSR(r)+2$. By Lemma 8.13, all processes will decide by the end of round $GSR(r)+4$. □

Lemma 8.15. *In every run r all correct processes decide by round $GSR(r)+5$.*

Proof. If some process correct process decides by the end of round $GSR(r)+2$, lets denote by k the round in which this happens, or $GSR(r)-1$ (the later round between the two). Since $k \geq GSR(r)-1$, in round $k+1$, it is assured that the decision message will reach $m+1$ processes, and in round $k+2$, it will reach all the process since each one receives a message from $n-m$ processes. Therefore, every process will decide by round $k+2$. If $k = GSR(r)+2$, $k+2 = GSR(r)+4$, and the lemma holds.

Suppose that no correct process decides by the end of round $GSR(r)+2$. If some process commits in round $GSR(r)$, , by Lemma 8.12, all processes adopt the same estimate by the end of round $GSR(r)+2$, and send it in round $GSR(r)+3$. By Lemma 8.13, all processes will decide by the end of round $GSR(r)+5$. If no process commits in $GSR(r)$, by Lemma 8.11, all processes will adopt the same estimate by the end of round $GSR(r)+1$, and send it in round $GSR(r)+2$. By Lemma 8.13, all processes will decide by the end of round $GSR(r)+4$. Note that if $GSR(r)=0$, no process can commit in round $GSR(r)$, and therefore decision occurs in five rounds (i.e., by $GSR(r)+5$). □

Chapter 9

Impossibility of Bounded Time Global Decision in $\Diamond MFM$

We define the $\Diamond$MFM family of models, for $m \in N^+, f \leq m < n/2$, as follows:

$\Diamond MFM(m)$ *(Majority-From-Majority)*: $t < n/2$, reliable links, every correct process is a $\Diamond(n-m)$-source and $\Diamond m$-accessible, m correct processes are $\Diamond n$-sources, and $(n-m)$ correct processes are $\Diamond(n-m)$-accessible.

Note that these models are only slightly weaker than $\Diamond$AFM, where we have shown that constant-time decision is attainable. We show that the time for global decision after GSR in all of these models is unbounded.

Lemma 9.1. *For any $m \in N^+$ s.t. $f \leq m < n/2$, there exists no consensus algorithm that reaches global decision in bounded time from GSR in $\Diamond$MFM(m).*

Proof. Assume by contradiction that an algorithm A reaches global decision by round GSR(r)+T_A in every run r. We partition the processes into three groups: a group P of m processes, a group Q of m processes, and a group R of the remaining $n-2m$ (≥ 1, since $m < n/2$) processes.

We construct three runs in $\Diamond MFM(m)$, in which no process fails ($f = 0$), and processes of each group have perpetually timely bidirectional links to all other processes of the same group. For each run we state which inter-group links are $\Diamond$timely. These links are timely only from GSR onward, and delay until round GSR all messages sent before that round.

Each one of the three runs is a run in $\Diamond MFM(m)$: in every run, either groups Q and R or groups P and R are fully connected with timely links from GSR onward. The number of processes in the resulting group is $n-m$. Therefore, in every run there are $n-m$ processes that are $\Diamond(n-m)$-accessible (and therefore $\Diamond(n-m)$-source and $\Diamond m$-accessible, since $n-m > m$). The other m processes are correct and fully interconnected with timely links from the start, i.e. m-accessible, and have $\Diamond$timely outgoing links to every process,i.e. m correct $\Diamond n$-source processes. Therefore, the requirements of the model are fulfilled in each one of the three runs below.

We construct a run σ_0 in which from round GSR(σ_0) $= 1$ onward (i) processes of P have timely outgoing links to all other processes, and (ii) processes of Q and R have timely links among them. All other links between groups deliver messages only after round GSR(σ_0)+$T_A = T_A + 1$. All processes propose 0. Since algorithm A always reaches global decision by round GSR(r)+T_A, and since GSR(σ_0) $= 1$, processes of P decide 0 (by validity) by round $T_A + 1$.

We next construct a run σ_1, which is identical to σ_0 until round $T_A + 1$, and from round GSR(σ_1) $= T_A + 2$ onward (i) processes of Q have timely outgoing links to all other processes, and (ii) processes of P and R have timely links among them. All other links between groups deliver messages only after round GSR(σ_1)+T_A. All processes propose 1. Since algorithm A always reaches global decision by round GSR(r)+T_A, processes of Q decide 1 (by validity) by round GSR(σ_1)+T_A.

Finally, we construct a run σ_2 which is identical to σ_1 (and σ_0) until round $T_A + 1$, and in which, like in σ_1, starting from round GSR(σ_2) $= T_A + 2$ onward (i) processes of Q have timely outgoing links to all other processes, and (ii) processes of P and R have timely links among them. All other links between groups deliver messages only after round GSR(σ_2)+T_A. In σ_2, processes of Q propose 1 and processes of P and R propose 0. Note that processes of Q decide 1 since they cannot distinguish this run from σ_1 - in both runs group Q is disconnected from other processes until GSR(σ_2)+T_A, and fully connected within. Therefore, the messages between processes in Q must be the same in both runs. Note that a run is fully determined by the initial states and the set of links that

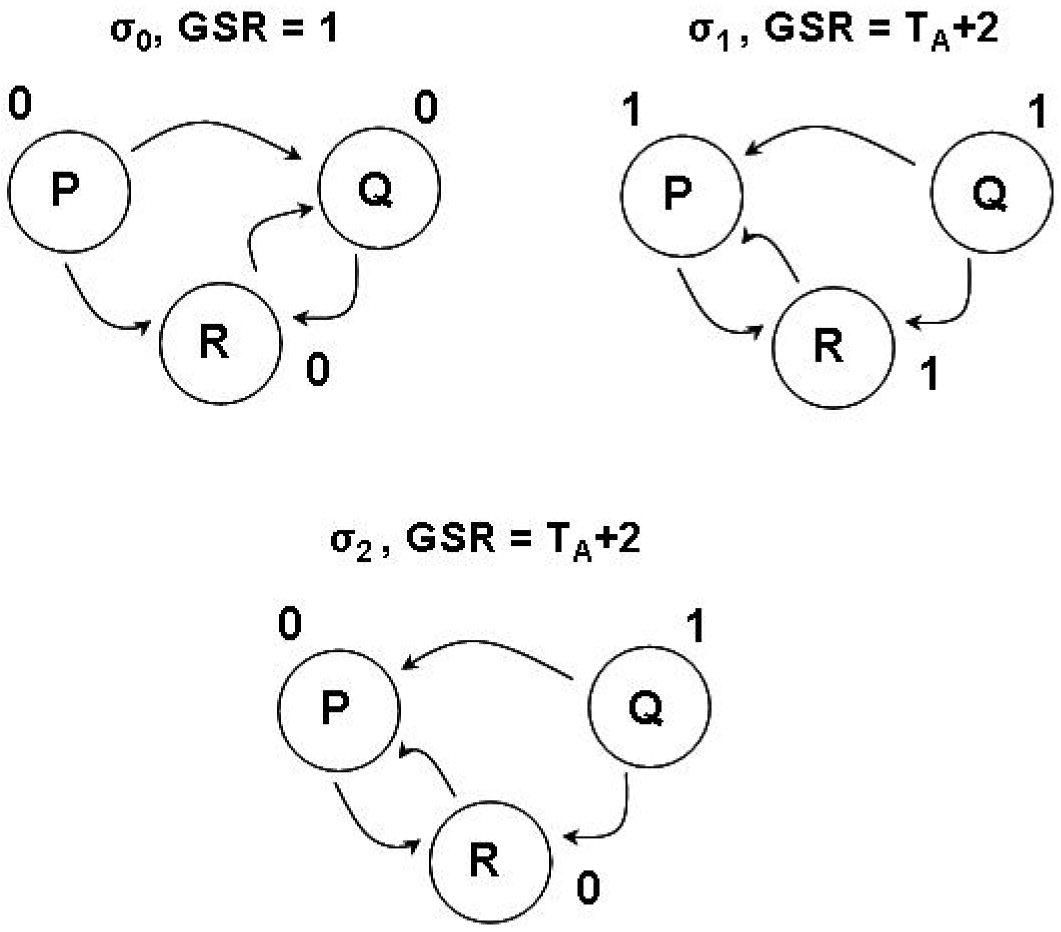

Figure 9.1: Illustration of the partition argument.

are timely in each round, because processing is deterministic, and all round messages are delivered together to the protocol in one end-of-round action (*compute()* does not see the order of arrival, only which set of messages was received in the round). Thus, processes of group P cannot distinguish σ_2 from σ_0 by round $T_A + 1$ and hence decide 0, violating agreement. A contradiction. □

Note that our notion of timely links is more abstract than the real-time-based definition used in [2, 3, 43], where messages arrive within bounded latency. Nevertheless, since we never explicitly reason about time duration in constructing our runs, our proof is applicable even if all messages on timely links in these runs are delivered within bounded latency, and hence covers these models.

Chapter 10

Probabilistic Comparison of Decision Time in Different Models

This thesis has dealt with the number of rounds needed, starting from GSR, to reach a global consensus decision in different models. It was already known that decision in Eventual Synchrony (ES) takes 3 rounds from GSR, and we showed an algorithm for the $\Diamond LM$ model that achieved termination in the same number of rounds, and an algorithm for the $\Diamond AFM$ model that incured a penalty of 2 (or 3, in some settings) rounds. We justified this increase in rounds after GSR by assessing that our consensus algorithms for $\Diamond AFM$ and $\Diamond LM$ will actually reach decision much faster than the algorithm for ES, even though the algorithm for $\Diamond AFM$ takes more rounds. Intuitively, GSR is reached faster in $\Diamond AFM$ and $\Diamond LM$ than in ES. We would like to probabilistically analyze this claim.

For this analysis, we model link failure probabilities as Independent and Identically Distributed (IID) Bernoulli random variables (like often done in the literature). By "link failure" we mean that the link fails to deliver a message in a timely manner, i.e. in the same round in which it was sent. In our calculations we make the simplification of not distinguishing the link of every process to itself as a special case, and consider it as any other link. We consider runs with $f = 0$, meaning that no process fails, and compare the Eventual Synchrony (ES), $\Diamond$AFM (with $n = 2m+1$) and $\Diamond LM$ models (with a predefined arbitrary leader).

All communication in some single round k can be represented as a n by n matrix A, where the rows are the destination process indices, the columns are the source process indices, and each entry $A_{i,j}$ is 0 if a message sent by p_j to p_i does not arrive in round k, and 1 if it does reach p_i in round k. Let p be the probability of any entry $A_{i,j}$ to be 1.

Recall that ES requires all entries in the matrix to be 1, for 3 consecutive rounds, $\Diamond AFM$ (with $n = 2m + 1$) requires the matrix to have a majority of 1's in every row and column, for 5 consecutive rounds. $\Diamond LM$ requires that for 3 consecutive rounds the matrix to have a majority of 1's in every row, such that every entry in the column corresponding to the leader is 1. We define random variables D_{ES}, $D_{\Diamond LM}$ and $D_{\Diamond AFM}$ to be the number of rounds until decision in the appropriate models.

10.1 Analysis of ES and $\Diamond LM$

The probability that A has 1 in every entry is: p^{n^2}. An optimal ES consensus algorithm reaches a global decision by round $GSR + 2$ in ES, thus we need the assumptions of ES to be satisfied for 3 consecutive rounds starting at some round $k \geq 1$. The probability of this to happen at any given round k is p^{3n^2}. Thus:

$$E(D_{ES}) = \frac{1}{p^{3n^2}} + 2 \tag{10.1}$$

Notice that for any fixed $p < 1$, $\lim_{n\to\infty} E(D_{ES}) = \infty$, since $\lim_{n\to\infty} p^{3n^2} = 0$.

For $\Diamond LM$, it is required that A has a majority of ones in all rows. If we assume that the leader is some fixed process p_k, we additionally require that $\forall 1 \leq j \leq n \ A_{j,k} = 1$. Denote the event that there is a majority of ones in row A_j by M and the event that $A_{j,k} = 1$ by L. Note that $Pr(L) = p$. By the multiplication rule we have:

$$Pr(L \cap M) = Pr(L) * Pr(M|L) = p \cdot Pr(M|L)$$

If the entry $A_{j,k}$ is 1, we are left with the $n - 1$ other entries in row A_j, out of which more than $\frac{n}{2} - 1$ entries must be 1. We get:

$$Pr(M|L) = \sum_{i=\lfloor \frac{n}{2} \rfloor}^{n-1} \binom{n-1}{i} p^i (1-p)^{n-1-i}$$

We have n rows (the rows are independent of each other), and global decision is achieved by round $GSR + 2$ in $\Diamond LM$, thus:

$$E(D_{\Diamond LM}) = \frac{1}{(Pr(L \cap M))^{3n}} + 2 = \frac{1}{(p \cdot \sum_{i=\lfloor \frac{n}{2} \rfloor}^{n-1} \binom{n-1}{i} p^i (1-p)^{n-1-i}))^{3n}} + 2 \quad (10.2)$$

For any fixed $p < 1$, its clear that $\lim_{n \to \infty} E(D_{\Diamond LM}) = \infty$, since $\lim_{n \to \infty} p^{3n} = 0$, and $Pr(M|L) \leq 1$. But $E(D_{\Diamond LM})$ grows much slower with n than $E(D_{ES})$, since the power of p is linear in n in $E(D_{\Diamond LM})$, and quadratic in n in $E(D_{ES})$.

10.2 Analysis of $\Diamond AFM$

Consider a given row k of A. We first analyze the probability that the row includes a majority of ones. To this end, let X_j be the random variable representing the cell $A_{k,j}$. According to our assumption, $X_1, X_2, ..., X_n$ are independent and identically distributed Bernoulli random variables with probability of success p. Let $X = \sum_{i=1}^{n} X_i$. The probability that any given row in A has a majority of 1's is:

$$Pr(X > \frac{n}{2}) = \sum_{i=\lfloor \frac{n}{2} \rfloor + 1}^{n} \binom{n}{i} p^i (1-p)^{n-i}$$

For n (independent) rows we get that the probability is $(Pr(X > \frac{n}{2}))^n$. Now assume that every row has a majority of 1 entries. The probability of any given entry to be 1 is still at least p. We therefore can make an identical calculation for the columns. To get a bound on the probability to have a majority of ones in each row and column, we therefore raise the expression again to the power of 2. Since the algorithm presented in this thesis for $\Diamond AFM$ with $n = 2m + 1$ (see Section 8.2) achieves global decision by round $GSR + 4$ (5 rounds from GSR), this needs to hold for 5 consecutive rounds, and therefore we will additionally raise the expression to the power of 5. We get:

$$E(D_{\Diamond AFM}) \leq \frac{1}{(\sum_{i=\lfloor \frac{n}{2} \rfloor +1}^{n} \binom{n}{i} p^i (1-p)^{n-i})^{10n}} + 4 \tag{10.3}$$

In the following lemma we show that, asymptotically, $E(D_{\Diamond AFM})$ approaches the constant value of 5 rounds, as n, the number of processes, goes to infinity.

Lemma 10.1. *For a fixed* $p > \frac{1}{2}$, $\lim_{n\to\infty} E(D_{\Diamond AFM}) = 5$

Proof. To bound the probability that A has a majority of 1's in a row, we use a Chernoff bound [14]: Let $X_1, X_2, ..., X_n$ and X be as defined above, and denote $\mu = E(X) = np$. By the Chernoff bound, for any $0 < \epsilon < 1$:

$$P(X \leq (1-\epsilon)\mu) < e^{-\mu\epsilon^2/2}$$

We would like to bound the probability $P(X \leq \frac{n}{2})$ and therefore take $\epsilon = (1 - \frac{1}{2p})$. Thus, for $p > 1/2$, we get:

$$P(X \leq \frac{n}{2}) \leq e^{-(1-\frac{1}{2p})^2 np/2}$$

and

$$P(X > \frac{n}{2}) > 1 - e^{-(1-\frac{1}{2p})^2 np/2}$$

This is a bound on the probability that any given row in A has a majority of 1's. For n (independent) rows, we get that the probability exceeds $(1 - e^{-(1-\frac{1}{2p})^2 np/2})^n$. As was already explained, if we take p as the lower bound for the probability that given a majority of ones in each row, any given entry in A is 1, we have to raise this expression to the power of 2. Additionally, this needs to hold for 5 consecutive rounds, and thus:

$$E(D_{\Diamond AFM}) \leq \frac{1}{(1 - e^{-(1-\frac{1}{2p})^2 np/2})^{10n}} + 4$$

For a fixed $p < 1$, the first expression in the sum above approaches 1 as $n \to \infty$, and therefore $E(D_{\Diamond AFM}) \to 5$. □

10.3 Numerical results

We construct graphs using formulas (10.1), (10.2) and (10.3), and compare E(D) - the expected number of rounds until global decision in ES, $\Diamond AFM$ (with $n = 2m + 1$) and $\Diamond LM$ (with some fixed process acting as the leader). As the formulas suggest, the graphs for ES and $\Diamond LM$ show the expected $E(D_{ES})$ and $E(D_{LM})$ round number, while the graphs for $\Diamond AFM$ show an upper bound on the expected $E(D_{AFM})$ round number. The graphs are shown either as a function of p, the probability of timely delivery on any single link, or as a function of n, the number of processes.

The number of processes used in our comparison is similar to the one used to analyze Petal [40] (a distributed virtual disks system) and Frangipani [48] (a distributed file system based on Petal), which employ Paxos and used up-to 8 servers to analyze system performance. Note that although round numbers are really integers, the graphs are interpolated and thus to get the real value, the graph value must be rounded towards the closest higher integer.

The graphs are presented in Figure 10.1. In Figure 10.1(a) we can see that even with a relatively small n and very high probability of timely message delivery, performance in ES deteriorates drastically as p decreases, while both $\Diamond AFM$ and $\Diamond LM$ maintain excellent performance. Figure 10.1(b) zooms in on the area of $p \geq 0.99$ in Figure 10.1(a), and again emphasizes the quick deterioration of the performance in ES, showing also the difference between the two other models in this high range of p - here $\Diamond LM$ performs better than $\Diamond AFM$. Figure 10.1(c) shows how the optimal algorithm in ES performs relatively to the algorithms in the two other models, as we fix p to be 0.99 and increase n. As was mentioned before, $\lim_{n\to\infty} E(D_{ES}) = \infty$ and this clearly shows on this graph, even for these small values of n (notice that $\Diamond LM$ deteriorates much slower, even though $\lim_{n\to\infty} E(D_{\Diamond LM}) = \infty$ as well).

Figure 10.1(d) compares only $\Diamond AFM$ and $\Diamond LM$ (as is clear from Figure 10.1(a), ES has extremely bad performance in this range of p). We can see that starting from about $p = 0.94$, $\Diamond LM$ and $\Diamond AFM$ have a very close performance, and as we saw in Figure 10.1(b), in higher ranges $\Diamond LM$ is even better than $\Diamond AFM$. As p gets smaller, $\Diamond LM$ performs much worse than $\Diamond AFM$. For example with $p = 0.8$, our $\Diamond AFM$ algorithm

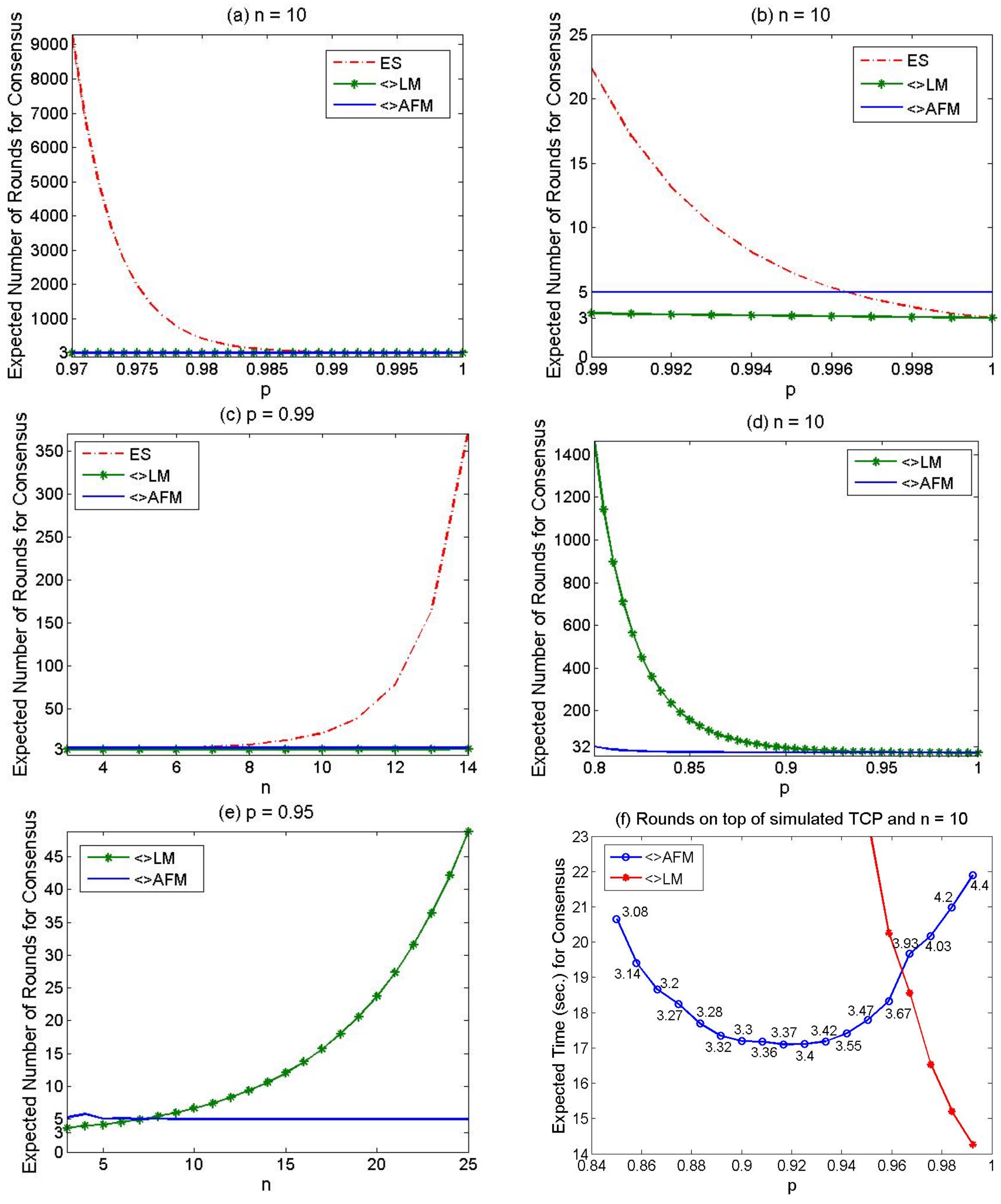

Figure 10.1: Probabilistic comparison of rounds/time until decision.

is expected to reach decision after about 32 rounds, while $\Diamond LM$ is expected to take more than 1400 rounds. Figure 10.1(e) shows that $\Diamond LM$ gets worse than $\Diamond AFM$ when we increase n as well (for a fixed p), while $\Diamond AFM$ keeps constant excellent performance. This is due to the asymptotic behavior of $E(D_{\Diamond LM})$ and $E(D_{\Diamond AFM})$ (discussed earlier in this chapter): while the first goes to ∞ when $n \to \infty$, the second approaches the value of 5 rounds (Lemma 10.1).

The results in this section show that our models are much easier to implement than ES (assuming IID), and that they should be preferred when messages might be late or lost and as the network size increases, since they will allow a much faster consensus decision. The intuition of why assuming ES will get such a bad performance, is that it is practically impossible to get 3 matrices not containing even a single zero entry, if the probability for a zero is non-negligible or the matrix is large enough. This is in contrast to the requirement of $\Diamond AFM$, which is almost always expected to be satisfied having a realistic probability of timely message delivery - for example, if a probability to get a 1 is $3/4$ and the matrix (the number of Bernoulli trials) is large enough, the ones are expected to be at least $3/4$ of every row and column, which is even more than $\Diamond AFM$ requires. $\Diamond LM$ is almost the same as $\Diamond AFM$ when p is high, but (somewhat surprisingly), there is a considerable difference in favor of $\Diamond AFM$ when p is low. This is because $\Diamond LM$ requires the messages from a single known process not to be lost for 3 consecutive rounds. We can see that the 2 round penalty of $\Diamond AFM$ is worthwhile, both comparing to ES and $\Diamond LM$, and that implementing the leader for $\Diamond LM$ is really not worth the effort, if the network is IID as we assumed in our analysis. We also conclude that the $\Diamond AFM$ model has the potential to greatly improve the scalability of distributed systems that use consensus.

10.4 Setting the Timeout in a Practical Scenario

We have shown that a higher p reduces the number of rounds for decision. On the other hand, it is obvious that in order to achieve a higher p, one needs larger timeouts, and therefore each individual round is longer. We wish to explore this tradeoff in a real practical setting. In this section, we examine one particular setting, using TCP latencies measured by Cardwell et al. [10], and determine the optimal timeout in this setting. Of course, this is but one example, and in different practical settings, the timeouts will be different.

Cardwell et al. [10] present a model of TCP latencies, validated by simulations. Specifically, we use the measured cumulative distribution of latencies in TCP simulations presented in Figure 7 in [10]. For example, in that experiment, 99% of the packets arrived within a latency of $4.5sec$, whereas 90% of packets arrived within $3.3sec$.

In Figure 10.1(f), we present our upper bound on the actual duration of the $\Diamond AFM$-based and $\Diamond LM$-based consensus algorithms, in seconds, in the setting of [10]. This duration is computed by multiplying the upper bound on the number of rounds required for decision with a given p from Equations (10.2) and (10.3), by the timeout required to achieve this p in the measurements of [10]. We take $n = 10$ processes, and $0.85 < p < 1$.

The graph of $\Diamond AFM$ in Figure 10.1(f) demonstrates the tradeoff: for $p < 0.92$ while the required number of rounds is increasing (as p gets smaller), the length of each round is decreasing. For $p > 0.92$ (as p gets larger) the number of required rounds decreases, but the cost of each round increases. For example - if we set our timeout to $4.03sec$, although the number of rounds will be almost minimal (approximately 5 rounds), the actual time until decision will be approximately $20.17sec$, which is about the same time we would get if we waited for only 85.5% of the messages although the required number of rounds would be higher. This shows that setting conservative timeouts (improving p) will not necessarily improve performance. As we see from this graph - it might actually make it worse. From Figure 10.1(f), we conclude that in this setting of [10], choosing the timeout to $3.37sec$ is optimal for the $\Diamond AFM$ algorithm.

It was shown in [9] and [7], that the maximal latency can be orders of magnitude longer than the average latency on a TCP link, and therefore, it is not feasible to assure that no messages are ever late ,i.e., to get $p = 1$. However, if we use $p = 0.99$ an ES-based algorithm is expected to take 23 rounds (see Figure 10.1(b)), i.e. $101.2sec$, which is much worse than the $\Diamond AFM$-based algorithm, that is expected to take only 5 rounds, i.e. $22sec$. Note that the $\Diamond LM$-based algorithm takes only about 3 rounds for $p = 0.99$ (see Figure 10.1(b)), which take $14.2sec$, and thus is better than $\Diamond AFM$ for this p. This will be the case for every $p \geq 0.97$, as the figure shows. Moreover, the $\Diamond LM$-based algorithm is expected to outperform the $\Diamond AFM$-based one, in another setting - when links are not IID, and one process (the leader) has much better outgoing links than others [6, 7].

Chapter 11

Conclusions and Future Directions

We have focused on the question of which timeliness or failure detector guarantees one should attempt to implement in a distributed system. While it is obvious that weaker timeliness/failure detector guarantees can be practically satisfied using shorter timeouts and cheaper hardware than stronger ones, it was not previously established what implications the use of weaker properties has on algorithm performance. Although from a theoretical perspective it is interesting to discover the weakest conditions that can be used to ensure *eventual* decision, in practice, timely decision is of essence. System designers are often willing to spend more on hardware, if this can ensure better performance. Likewise, implementations are better off using longer timeouts if this can lead to faster decision overall.

We have presented a general framework GIRAF, to answer such questions. GIRAF does not restrict the set of allowed algorithms, models or failure patterns, but rather organizes algorithms in a "round" structure, which allows for analyzing their complexity. We used our framework to show that some previously suggested guarantees were too weak to solve consensus in a timely manner. We have further shown that it is possible to strengthen a model in which consensus is not solvable in bounded time ($\Diamond$MFM(m) for $n = 2m+1$) to get a model in which consensus is solvable in constant time ($\Diamond$AFM) by adding just one $\Diamond$timely incoming link per process, for a minority of processes. In such situations, it is worthwhile to increase timeouts and/or buy faster hardware in order to implement stronger guarantees. On the other hand, we have shown that the strong ES model (which requires timely communication among *all* pairs of correct processes) can be weakened in ways that are significant from a performance standpoint (as shown in Chapter 10 and [6, 7]), and yet with little (for $\Diamond$AFM) or no (for $\Diamond$LM) penalty on the number of communication rounds employed by the consensus algorithm. In fact, our probabilistic analysis (Chapter 10) has shown that $\Diamond AFM$ is extremely scalable, significantly more than ES or even $\Diamond LM$.

We believe that GIRAF has the potential to further enhance the understanding of performance tradeoffs between different models, and opens vast opportunities for future work. We now point out several exemplar directions for future research.

- One can use our new notion of α-reducibility (and k-round reducibility) to compare various models more meaningfully than with the classical notion of reducibility, by considering the time (round) complexity of the reduction.

- While this thesis focuses on the performance of the algorithm after synchronization, an important complementary direction for future study is understanding the performance of the environment's synchronization mechanism, that is, the actual time it takes to reach GSR in various timing models. Whereas GIRAF provides generic analysis of the cost of algorithms in terms of different round-types (e.g., all-to-all communication in each round or communication with a majority of processes), in order to deduce which algorithm is best for a given network setting, this analysis should be complemented with a measurement study of the cost of rounds of different types in that specific setting.

- It would be interesting to further study the fine line between models that allow bounded and unbounded decision times. For example, is it possible to weaken $\Diamond$AFM by making fewer processes $\Diamond(m+1)$-sources, and still achieve constant or bounded time consensus? and what would be the effect of weakening the assumption that the leader is a $\Diamond$n-source in $\Diamond$LM, on consensus performance?

- In this thesis, we have focused on global decision. It can be interesting to investigate local consensus decision [25], i.e., the number of rounds until *some* process decides.

- Finally, there are gaps between upper and lower bounds shown in Table 1, which might be closed.

Bibliography

[1] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg. Stable leader election. In *DISC*, pages 108–122, 2001.

[2] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg. On implementing omega with weak reliability and synchrony assumptions. In *PODC*, pages 306–314, 2003.

[3] M. K. Aguilera, C. Delporte-Gallet, H. Fauconnier, and S. Toueg. Communication-efficient leader election and consensus with limited link synchrony. In *PODC*, pages 328–337, 2004.

[4] E. Anceaume, A. Fernandez, A. Mostefaoui, G. Neiger, and M. Raynal. A necessary and sufficient condition for transforming limited accuracy failure detectors. *J. Comput. Syst. Sci.*, 68(1):123–133, 2004.

[5] H. Attiya and J. L. Welch. *Distributed computing: fundamentals, simulations and advanced topics*. McGraw-Hill, Inc., Hightstown, NJ, USA, 1998.

[6] O. Bakr. Performance evaluation of distributed algorithms over the Internet. Master's thesis, Massachusetts Institute of Technology, Feb. 03.

[7] O. Bakr and I. Keidar. Evaluating the running time of a communication round over the Internet. In *PODC*, pages 243–252, 2002.

[8] R. A. Bazzi and G. Neiger. Simplifying fault-tolerance: providing the abstraction of crash failures. *J. ACM*, 48(3):499–554, 2001.

[9] N. Cardwell, S. Savage, and T. Anderson. Modeling the performance of short tcp connections, 1998.

[10] N. Cardwell, S. Savage, and T. Anderson. Modeling tcp latency. In *INFOCOM*, pages 1742–1751, 2000.

[11] T. D. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving consensus. *J. ACM*, 43(4):685–722, July 1996.

[12] T. D. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *J. ACM*, 43(2):225–267, 1996.

[13] B. Charron-Bost and A. Schiper. Uniform consensus is harder than consensus. *J. Algorithms*, 51(1):15–37, 2004.

[14] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on a sum of observations. *Ann. Math. Statist.*, 23:493–507, 1952.

[15] G. Chockler, I. Keidar, and R. Vitenberg. Group communication specifications: a comprehensive study. *ACM Computing Surveys*, 33(4):427–469, 2001.

[16] F. Chu. Reducing Ω to $\diamond W$. *Information Processing Letters*, 67(6):289–293, Sept. 1998.

[17] F. Cristian and C. Fetzer. The timed asynchronous distributed system model. In *28th Annual Intl. Symp. on Fault-Tolerant Computing*, June 1998.

[18] F. Cristian and C. Fetzer. The timed asynchronous distributed system model. In *IEEE Transactions on Parallel and Distributed Systems*, number 10(6), pages 642–657, June 1999.

[19] D. Dobre and N. Suri. One step consensus with zero degradation. In *DSN*, 2006.

[20] P. Dutha, R. Guerraoui, and I. Keidar. The overhead of consensus failure recovery. Technical Report 200456, École Polytechnique Fédérale de Lausanne, 2004.

[21] P. Dutha, R. Guerraoui, and I. Keidar. The overhead of consensus failure recovery. Submitted for publication, 2005.

[22] P. Dutta and R. Guerraoui. Fast indulgent consensus with zero degradation. In *Fourth European Dependable Computing Conference (EDCC-4)*, Oct. 2002.

[23] P. Dutta and R. Guerraoui. The inherent price of indulgence. In *21st ACM Symp. on Principles of Distributed Computing (PODC-21)*, July 2002.

[24] P. Dutta, R. Guerraoui, and L. Lamport. How fast can eventual synchrony lead to consensus?. In *DSN*, pages 22–27, 2005.

[25] P. Dutta, R. Guerraoui, and B. Pochon. Tight lower bounds on early local decisions in uniform consensus. In *17th Intl. Symp. on Distributed Computing (DISC-17)*, pages 264–278, Oct 2003.

[26] C. Dwork, N. A. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, Apr. 1988.

[27] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32(2):374–382, Apr. 1985.

[28] E. Gafni. Round-by-round fault detectors: Unifying synchrony and asynchrony. In *17th ACM Symp. on Principles of Distributed Computing (PODC-17)*, pages 143–152, 1998.

[29] R. Guerraoui. Revisiting the relationship between non blocking atomic commitment and consensus problems. In *9th Intl. Wshop on Distributed Algorithms (WDAG-9)*, number 791 in Lecture Notes in Computer Science, pages 87–100. Springer-Verlag, Sept. 1995.

[30] R. Guerraoui. Indulgent algorithms. In *19th ACM Symp. on Principles of Distributed Computing (PODC-19)*, pages 289–298, July 2000.

[31] R. Guerraoui and M. Raynal. The information structure of indulgent consensus. *IEEE Transactions on Computers*, 53(4):453–466, 2004.

[32] R. Guerraoui and A. Schiper. ”Γ-accurate” failure detectors. In *WDAG*, pages 269–286, 1996.

[33] I. Keidar and S. Rajsbaum. On the cost of fault-tolerant consensus when there are no faults - a tutorial. Technical Report MIT-LCS-TR-821, MIT, May 2001.

[34] I. Keidar and S. Rajsbaum. A simple proof of the uniform consensus synchronous lower bound. 2002.

[35] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

[36] L. Lamport. The part-time parliament. Technical Report 49, Systems Research Center, Digital Equipment Corp, Palo Alto, Sept. 1989. A revised version of the paper also appeared in ACM Transaction on Computer Systems, 16(2):133-169, May 1998.

[37] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

[38] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[39] M. Larrea, S. Arevalo, and A. Fernandez. Efficient algorithms to implement unreliable failure detectors in partially synchronous systems. In *International Symposium on Distributed Computing*, pages 34–48, 1999.

[40] E. K. Lee and C. A. Thekkath. Petal: Distributed virtual disks. In *ASPLOS*, pages 84–92, 1996.

[41] N. Lynch and M. Tuttle. An introduction to Input/Output Automata. *CWI Quarterly*, 2(3):219–246, 1989.

[42] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[43] D. Malkhi, F. Oprea, and L. Zhou. Omega meets paxos: Leader election and stability without eventual timely links. *19th Intl. Symp. on Distributed Computing (DISC-19)*, pages 199–213, sep 2005.

[44] A. Mostefaoui and M. Raynal. Solving consensus using Chandra-Toueg's unreliable failure detectors: A general quorum-based approach. In *13th Intl. Symp. on Distributed Computing*, pages 49–63, Sept. 1999.

[45] G. Neiger and S. Toueg. Automatically increasing the fault-tolerance of distributed algorithms. *J. Algorithms*, 11(3):374–419, 1990.

[46] N. Santoro and P. Widmayer. Time is not a healer. *6th Annual Symp. Theor. Aspects of Computer Science*, volume 349 of LNCS:304–313, feb 1989.

[47] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Comput. Surv.*, 22(4):299–319, Dec. 1990.

[48] C. A. Thekkath, T. Mann, and E. K. Lee. Frangipani: a scalable distributed file system. In *SOSP '97: Proceedings of the sixteenth ACM symposium on Operating systems principles*, pages 224–237, New York, NY, USA, 1997. ACM Press.