Replication predicates for dependent-failure algorithms

Flavio Junqueira and Keith Marzullo

Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA – USA
{flavio, marzullo}@cs.ucsd.edu

Abstract. *To establish lower bounds on the amount of replication, there is a common partition argument used to construct indistinguishable executions such that one violates some property of interest. This violation leads to the conclusion that the lower bound on process replication is of the form $n > \lfloor kt/b \rfloor$, where t is the maximum number of process failures in any of these executions and k, b are positive integers. In this paper, we show how this argument can be extended to give bounds on replication when failures are dependent. We express these bounds in terms of our model of* cores *and* survivors sets *using set properties instead of predicates of the form $n > \lfloor kt/b \rfloor$. We give two different properties that express the same requirement for $k > 1$ and $b = 1$. One property comes directly from the argument, and the other is more useful when designing an algorithm that takes advantage of dependent failures. We also consider a somewhat unusual replication bound of $n > \lfloor 3t/2 \rfloor$ that arises in the Leader Election problem for synchronous receive-omission failures. We generalize the replication bound for dependent failures, and develop an algorithm that shows that this generalized replication bound is tight.*

1 Introduction

Lower bounds for the amount of process replication are often arrived at by an argument of the following flavor:

1. Partition the n processes into k blocks, where each block has at most $\lceil t/b \rceil$ processes, $t \geq \lceil nb/k \rceil$, and k, b are positive integers such that $k > b \geq 1$.
2. Construct a set of executions. For each block A, there is at least one of the executions in which all the processes in A are faulty.
3. Given the set of executions, show that some property of interest is violated. Conclude that if the maximum number of faulty processes in an execution is never larger than t, then $t < \lceil nb/k \rceil$ and $n > \lfloor kt/b \rfloor$.[1]

Examples of such proofs include Consensus with arbitrary processor failures and no digital signatures requiring $n > 3t$ ($k = 3$, $b = 1$) [20], Primary Backup with general omission failures requiring $n > 2t$ ($k = 2$, $b = 1$) [23], and Consensus with the eventually strong failure detector $\diamond S$ requiring $n > 2t$ ($k = 2$, $b = 1$) [6,7].

We call a predicate like $n > \lfloor kt/b \rfloor$ a *replication predicate*: it gives a lower bound on the number of processes that are required given all possible sets of faulty processes.

Expressing bounds in terms of t is often referred to as a *threshold model*. Using t to express the number of faulty processes is convenient, but the bounds can lead to mistaken conclusions when processes do not fail independently or do not have identical probabilities of failure. This is because one is assuming that *any* subset of t or fewer processes can be faulty, which implies that failures are independent and identically distributed (IID). To use an algorithm developed under the threshold model on a system that does not have IID failures, one can compute the maximum number of processes that can fail in any execution, and then use that number as t. On the other hand one may be able to use fewer processes if an algorithm based on non-IID failures is used instead.

In an earlier paper we introduced a method for modeling non-IID failures [16] and studied Consensus under this model. We derived replication requirements in our new model and presented

[1] Some authors have used different symbols, such as f, to indicate an upper bound on the number of faulty processes.

protocols that showed these bounds to be tight. This paper generalizes the results of our earlier paper to protocols other than Consensus. We show how the lower bound argument given above can be easily generalized to accommodate our model of non-IID failures. This argument leads to a replication predicate that we call k–Partition, which generalizes the replication predicate $n > kt$ $(b = 1)$ for when failures are not IID.

The k–Partition property, however, may not prove to be very useful when designing an algorithm. An equivalent property, which we call k–Intersection, is often more useful for this purpose. It is more useful for designing algorithms because algorithms often refer to minimal sets of correct processes ($n - t$ processes when process failures are IID). These properties generalize the two properties we developed for Consensus in our earlier paper.

In this paper, after reviewing our failure model, we define the replication predicate k–Partition for $k > 1$. We then define k–Intersection and show that it is equivalent to k–Partition. We illustrate the utility of k–Intersection by showing that the *M-Availability* property [21] for Byzantine Quorum Systems is equivalent to 4–Intersection. Thus, a system that requires M-Availability has a replication predicate of 4–Partition.

Finally, we examine one point in the space of replication predicates for $b > 1$. We do so by considering a weak version of the *Leader Election* problem for synchronous systems that can suffer receive-omission failures. We review a previously-given lower bound proof that argues $n > \lfloor 3t/2 \rfloor$ $(k = 3, b = 2)$ for IID failures. This proof yields a definition that we call (3,2)–Partition. We derive an equivalent (3,2)–Intersection property and use it to develop an optimal protocol for Weak Leader Election. An immediate consequence is that the lower bound $n > \lfloor 3t/2 \rfloor$ for IID failures is tight. We believe that this is the first time this has been shown.

2 System model

We assume a system that is amenable to the lower bound proof described in the previous section. Such systems are often comprised of processes that communicate with messages. We consider systems in which processes can be faulty (as compared, for example, to systems in which the failure of messages to be delivered are attributed to faulty links rather than omission failures of processes).

Our work is based on a model of non-independent, non-identically distributed failures. We characterize failure scenarios with *cores* and *survivor sets* [15,16]. A core is a minimal subset of processes such that, in every execution, there is at least one process in the core that is not faulty. A core generalizes the idea of a subset of processes of size $t + 1$ in the threshold model. A *survivor set* is a minimal set of processes such that there is an execution in which none of the processes in the set are faulty. A survivor set generalizes the idea of a subset of processes of size $n - t$ in the threshold model, where n is the total number of processes. Cores and survivor sets are duals of each other: from the set of cores one can obtain the set of survivor sets by finding all minimal subsets of processes that intersect every core.

More formally, we define cores and survivor sets as follows. Consider a system with a set $\Pi = \{p_1, p_2, \ldots, p_n\}$ of processes. Let Φ be all of the executions of a distributed algorithm *alg* run by the processes in Π, and let $Correct(\phi)$ be the set of processes that are not faulty in an execution $\phi \in \Phi$.

Definition 1. *A subset $C \subseteq \Pi$ is a core if and only if:*

1. $\forall \phi \in \Phi,\ Correct(\phi) \cap C \neq \emptyset$;
2. $\forall p_i \in C,\ \exists \phi \in \Phi$ *such that* $C \setminus \{p_i\} \cap Correct(\phi) = \emptyset$.

Definition 2. *A subset $S \subseteq \Pi$ is a survivor set if and only if:*

1. $\exists \phi \in \Phi,\ Correct(\phi) = S$;
2. $\forall \phi \in \Phi, p_i \in S,\ Correct(\phi) \not\subseteq S \setminus \{p_i\}$.

In [16], we defined cores and survivor sets using probabilities. In this paper, we use an alternative definition, based on executions, that is more convenient when discussing algorithms. In practice, one can use failure probabilities and a target reliability (or availability) to compute the sets of faulty

processes that can be tolerated, and these sets determine the possible failures of an execution. However, one does not have to determine tolerated sets of faulty processes on probabilities. As our example below show, it can be based on a combination of quantitative and qualitative information.

We use the term *system profile* to denote a description of the tolerated failure scenarios. In the threshold model, a system profile is a pair $\langle \Pi, t \rangle$, which means that any subset of t processes in Π can be faulty. In our dependent failure model, the system profile is a triple $\langle \Pi, \mathcal{C}_\Pi, \mathcal{S}_\Pi \rangle$, where $\mathcal{C}_\Pi$ is the set of cores and $\mathcal{S}_\Pi$ is the set of survivor sets.[2] We assume that each process is a member of at least one survivor set (otherwise, that process can be faulty in each execution, and ignored by the other processes), and that no process is a member of every survivor set (otherwise, that process is never faulty). The threshold system profile $\langle \Pi, t \rangle$ is equivalent to the profile $\langle \Pi, \mathcal{C}_\Pi, \mathcal{S}_\Pi \rangle$ where $\mathcal{C}_\Pi$ is all subsets of Π of size $t+1$ and $\mathcal{S}_\Pi$ is all subsets of Π of size $|\Pi| - t$.

We treat the *kind* of failure—crash, omission, arbitrary, etc.—as a separate part of the failure model. The kind of failure is important both in the design of algorithms and in the derivation of lower bounds. In some situations, such as with hybrid failure models (for example, [8]), separating the kind of failures from the system profile would be complex. In general, we do not assume any particular kind of failure, but we do so when discussing specific problems.

Determining the system profile requires one to consider the possible causes of processes failures. For example, a process running on a particular processor fails if the processor hardware fails (crash failure). As another example, if one is concerned about software faults (bugs), then a process can fail if there is an error in one of the software packages it depends upon, and the system executes the erroneous instructions (which can result in an arbitrary failure) [11].

2.1 Determining a system profile

We now give an example of a system profile that uses qualitative information. In the work by Castro *et al.* [5], the authors observe that independent software development ideally produces disjoint sets of software faults. This observation is the basic idea of n-version programming, whose goal is to render software failures independent. Of course, there is still a marginal probability that two or more replicas fail in the same execution, but this probability is assumed to be small enough so that it can be ignored.

Suppose we want to implement a fault-tolerant service using the State Machine approach [4,25], and we are concerned about arbitrary failures arising from software faults. Moreover, we want to leverage the existence of multiple standalone implementations of this service we are interested in, as in BASE [5]. Thus, each replica has two components: a standalone implementation and a replica-coordination component that implements a distributed Consensus algorithm.

For this particular service, suppose that there are five standalone versions available: v_1 through v_5. Looking more carefully at the history of these versions, we discover that two of them reuse code from previous versions. More specifically, v_2 reuses a set X of modules from v_1, and v_3 reuses modules Y from v_1 and Z from v_2.[3] We also assume that X, Y, and Z are disjoint sets, and that v_4 and v_5 were developed independently.

Assuming that every software module potentially has software faults, we have: 1) faults in the modules in X can affect both v_1 and v_2; 2) faults in the modules in Y can affect both v_1 and v_3; 3) faults in the modules in Z can affect both v_2 and v_3.

Consider a system in which there is at least one replica running each of the five versions. Let E_i be the set of executions in which at least one replica is faulty because of a fault in the version v_i. These sets of executions are related to each other as shown in Figure 1.

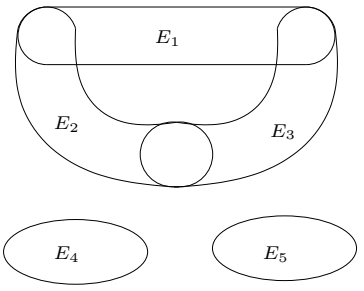

Fig. 1: E_i is the set of executions that have at least one faulty replica running version v_i

Assuming one replica for each version, and assuming that at most one software fault can be exercised in an execution, then we have the following system profile:

[2] Since $\mathcal{C}_\Pi$ and $\mathcal{S}_\Pi$ can be computed from each other, in fact the system profile could contain only one of these two sets. We include both for convenience.

[3] A module is a collection of functions and data structures.

Example 1.

$\Pi = \{p_1, p_2, p_3, p_4, p_5\}$
$\mathcal{C}_\Pi = \{\{p_1, p_2, p_3\}, \{p_4, p_5\}\} \cup \{\{p_i, p_j\} : i \in \{1, 2, 3\} \wedge j \in \{4, 5\}\}$
$\mathcal{S}_\Pi = \{\{p_i, p_4, p_5\} : i \in \{1, 2, 3\}\} \cup \{\{p_1, p_2, p_3, p_i\} : i \in \{4, 5\}\}$

This system has sufficient replication to implement Consensus in a synchronous system with arbitrarily faulty processes and no digital signatures [16]. The amount of replication is also sufficient to implement a fault-tolerant state machine for arbitrarily faulty processes using PBFT [4].

PBFT is an attractive protocol because it assumes a weak failure model. It was designed, however, assuming a threshold failure model. In the system profile given above, the smallest survivor set has three processes, which means that there are executions in which two processes are faulty. Hence, there is *not* enough replication to run PBFT: seven processes are required to tolerate two faulty processes. PBFT can be used by having one process execute v_1, one process execute v_2, one process executes v_3, two processes execute v_4, and two processes execute v_5. It is easy to check that there is no more than two failures in any execution of this configuration. Alternatively, we can implement a replica coordination component with a modified version of PBFT that can be run in the five process system of the example. In this case, the PBFT implementation needs to know the system profile in the same way that an unmodified PBFT (one assuming a threshold) needs to know the maximum number of faulty processes in an execution.[4]

This example illustrates an important point about dependent failures. Since IID failures can be represented as a particular system profile, lower bound proofs that hold for IID failures also hold in our model. But, if one has a system in which failures are not IID, then one should use an algorithm that explicitly uses a system profile. By using such an algorithm, it is often possible to use less replication than it requires when using an algorithm developed using the threshold model.

2.2 Relating survivor sets to fail-prone systems and adversary structures

We are not the first to consider non-IID behaviors: quorum systems have addressed the issues of non-IID behavior for some time. In [21], the idea of *fail-prone systems* was introduced. This paper gives the following definition for a set of servers U:

> A *fail-prone system* $\mathcal{B} \subseteq 2^U$ is a non-empty set of subsets of U, none of which is contained in another, such that some $B \in \mathcal{B}$ contains all the faulty servers.

This paper then observes that a fail-prone system can be used to generalize to less uniform assumptions than a typical threshold assumption. Their definition does not give a name to the elements of $\mathcal{B}$; we call each one a *fail-prone set*. As fail-prone sets are maximal, a fail-prone set is the complement of a survivor set and $\mathcal{B} = \{\Pi \setminus S : S \in \mathcal{S}_\Pi\}$. Although both survivor sets and fail-prone sets characterize failure scenarios, survivor sets have a fundamental use: if a process is collecting messages from the other processes, it can be fruitless to wait for messages from a set larger than a survivor set. Of course, there are times when fail-prone sets are more useful. For example, if $B_{\max}$ is the largest fail-prone set, then $|B_{\max}|$ is the value of t to use if one wishes to use a threshold-based protocol.

Non-threshold protocols were also considered in the context of secure multi-party computation with adversary structures [1,13,18]. Adversary structures are similar to fail-prone systems. They differ in two ways. First, adversary structures can represent more than one failure mode, e.g., crash failures and arbitrary failures. Each failure mode is described with sets of possibly faulty processes (processes are referred to as *players* in this literature). Second, the sets of possibly faulty players given in an adversary structure are not necessarily maximal; all sets of possibly faulty players are given. Using all possible sets of players that can deviate from the correct protocol behavior as opposed to only maximal sets (or minimal sets of correct processes, as with survivor sets) gives one more expressiveness in modeling system failures. Using fail-prone systems or survivor sets, however, is sufficient for establishing the bounds on process replication we show in this paper. Moreover,

[4] Although the original PBFT algorithm assumes a threshold on the number of failures, it is possible to modify it to work with cores and survivor sets. A discussion of these modifications, however, is outside of the scope of this paper.

these bounds hold even for a more expressive model such as adversary structures. This is because we use properties about the intersections of sets of correct processes. If the intersection property holds for some sets of processes A_1, A_2, ..., A_m then it holds for the sets of processes $A'_1 \supset A_1$, $A'_2 \supset A_2$, ..., $A'_m \supset A_m$. Hence, one only has to consider the minimal sets of correct processes in these intersection properties. The more expressive failure models have no benefit when establishing bounds on process replication.

3 k properties

In the generic lower bound proof described in Section 1, one first partitions the set of processes into k blocks, and then constructs a set of executions. For each block A, there is at least one execution in which all the processes in A are faulty. Being able to fail all the processes of a particular block then enables the construction of an execution in which some property is violated. For example, for Consensus, the property violated is *agreement*. For Primary-Backup algorithms, the property violated is the one that says that at any time there is at most one primary.

Having derived a contradiction, the proof concludes by stating that one cannot partition the processes in the manner that was done. With the threshold model and $b = 1$, this implies that not all processes of any subset of size $\lceil n/k \rceil$ can be faulty, and consequently $t < \lceil n/k \rceil$. In our dependent failure model, this implies that in any partition of the processes into k blocks, there is at least one block A that does not contain only faulty processes: A contains a core. More formally, let $\mathcal{P}_k(\Pi)$ be the set of partitions of Π into k blocks. We then have the following property for a system profile $\langle \Pi, \mathcal{C}_\Pi, \mathcal{S}_\Pi \rangle$:

Property 1. k–**Partition**, $k > 1$, $|\Pi| > k$: $\forall \mathcal{A} \in \mathcal{P}_k(\Pi) : \exists A_i \in \mathcal{A} : \exists C \in \mathcal{C}_\Pi : C \subseteq A_i$

Although k–Partition is useful for lower bound proofs, it is often not very useful for the design of algorithms. Survivor sets are often more convenient to refer to than cores. For example, the algorithm for Consensus by Chandra and Toueg for crash failures in asynchronous systems with failure detectors of the class $\diamond S$ assumes at least $2t + 1$ processes. For this number of processes, any pair of subsets of size $n - t$ has a non-empty intersection, and this property is crucial to avoid the violation of agreement. This is equivalent to stating that any two survivor sets intersect, or equivalently that $\mathcal{S}_\Pi$ is a coterie [10]. In the algorithm for Interactive Consistency by Lamport *et al.* [20], at least $3t + 1$ processes are required, and for this number of processes any intersection between a pair of subsets of size $n - t$ contains at least $t + 1$ processes. Consequently, every intersection between such subsets contains at least one correct process. This is equivalent to saying that the intersection of any two survivor sets contains a core, or equivalently that the intersection of any three survivor sets is not empty.[5]

3.1 k–Intersection

We now state the property that we show to be equivalent to k–Partition and that references survivor sets instead of cores. We call it k–Intersection. k–Intersection states that for a system profile $\langle \Pi, \mathcal{C}_\Pi, \mathcal{S}_\Pi \rangle$, for every set $T \subset \mathcal{S}_\Pi$ of size k, there is some process that is in every element of T. Let $\mathcal{G}_x(A)$ be the set of all the subsets of A of size x; if $|A| < x$, then $\mathcal{G}_x(A) = \emptyset$. We have the following property for a system profile $\langle \Pi, \mathcal{C}_\Pi, \mathcal{S}_\Pi \rangle$:

Property 2. k–**Intersection**, $k > 1$, $|\Pi| > k$, $|\mathcal{S}_\Pi| > k$: $\forall T \in \mathcal{G}_k(\mathcal{S}_\Pi) : (\cap_{S \in T} S) \neq \emptyset$

As an illustration, we have that the set $\mathcal{S}_\Pi$ in Example 1 satisfies 3–Intersection.

We now show the equivalence between k–Partition and k–Intersection. For both directions, we prove by contrapositive. To show that k–Partition implies k–Intersection, we assume that k–Intersection does not hold (there are k survivor sets that do not intersect), and build a partition

[5] Recall that a survivor set is a minimal subset of Π that intersects every $C \in \mathcal{C}_\Pi$. Equivalently, a core is a minimal subset of Π that intersects every $S \in \mathcal{S}_\Pi$. Hence, if the intersection of two survivor sets contains a core, it also intersects every survivor set. And, if the intersection of two survivor sets intersects every other survivor set, then the intersection contains a core.

that violates k–Partition. This partition is such that every block of the partition does not contain elements from some survivor set. For the other direction, we assume that there is a partition of Π into k blocks such that no block contains a core. If a block does not contain a core, then the union of the remaining $k-1$ blocks must contain a survivor set. We use this argument to show that there is a set of at most k survivor sets that do not intersect, thus violating k–Intersection.

Theorem 1. *k–Partition $\equiv$ k–Intersection*

Proof. $\Rightarrow$: Proof by contrapositive. Suppose a system profile $\langle \Pi, \mathcal{C}_\Pi, \mathcal{S}_\Pi \rangle$ such that there is a subset $\mathcal{S} = \{S_1, \ldots, S_k\} \subset \mathcal{S}_\Pi$ such that $\bigcap \mathcal{S} = \emptyset$. We then build a partition $\mathcal{A} = \{A_1, \ldots, A_k\}$ as follows:

$$
\begin{aligned}
A_1 &= \Pi \setminus S_1 \\
A_2 &= \Pi \setminus (S_2 \cup A_1) \\
&\vdots \\
A_i &= \Pi \setminus (S_i \cup A_1 \cup A_2 \ldots \cup A_{i-1}) \\
&\vdots \\
A_k &= \Pi \setminus (S_k \cup A_1 \ldots \cup A_{k-1})
\end{aligned}
$$

Suppose without loss of generality that no A_i is empty. It is clear from the construction that no two distinct blocks A_i, A_j intersect. It remains to show that: 1) $\bigcup \mathcal{A} = \Pi$; 2) $\forall i \in \{1, \ldots, k\} : A_i$ does not contain a core. To show 1), consider the following derivation:

$$
\begin{aligned}
\bigcup \mathcal{A} &= (\Pi \setminus S_1) \cup (\Pi \setminus (S_2 \cup A_1)) \cup \ldots \cup (\Pi \setminus (S_k \cup A_1 \cup A_2 \ldots \cup A_{k-1})) && (1) \\
&= \Pi \setminus ((S_1 \cap (S_2 \cup A_1)) \cap \ldots \cap (S_k \cup A_1 \cup A_2 \ldots \cup A_{k-1})) && (2) \\
&= \Pi \setminus (S_1 \cap S_2 \cap \ldots \cap (S_k \cup A_1 \cup A_2 \ldots \cup A_{k-1})) && (3) \\
&\vdots \\
&= \Pi \setminus (\cap_i S_i) && (4) \\
&= \Pi && (5)
\end{aligned}
$$

Explaining the derivation:

- Line 1 to Line 2 follows from the observation that for any subsets A, B of Π, we have that $(\Pi \setminus A) \cup (\Pi \setminus B) = \Pi \setminus (A \cap B)$;
- Line 2 to Line 3: the intersection between S_1 and A_1 has to be empty, since S_1 contains exactly the elements we removed from Π to form A_1;
- Line 3 to Line 4: by repeating inductively the process used to derive Line 3, we remove every term A_i present in the equation;
- Line 4 to Line 5: By assumption, the intersection of S_1 through S_k is empty.

To show 2), we just need to observe that any A_i is such that we removed all the elements of S_i. By the definitions of a core and of a survivor set, a subset that does not contain elements from some survivor set does not contain a core.

$\Leftarrow$: Proof also by contrapositive. Let $\langle \Pi, \mathcal{C}_\Pi, \mathcal{S}_\Pi \rangle$ be a system profile such that there is a partition $\{A_1, \ldots, A_k\}$ of Π in which no A_i contains a core. Because no block contains elements from every survivor set (no block contains a core), we have that for every A_i, there is a survivor set S_i such that $S_i \cap A_i = \emptyset$. Consequently, we have that $\cap_i S_i$ is empty, otherwise either some A_i contains an element that is in $\bigcap_i S_i$ or $\{A_1, \ldots, A_k\}$ is not a partition, either way contradicting our previous assumptions.

In the remainder of this section, we discuss the utility of these properties. In particular, we show the equivalence between 4–Intersection and M-Consistency. M-Consistency is a property that a set of quorums have to satisfy to mask Byzantine failures [21].

3.2 4–Intersection and M-Consistency

In [21], the following *M-Consistency* property was defined. It was stated that this property was necessary for one to implement a Masking Byzantine Quorum System. This property allows a process to identify a result from a non-faulty server. The set $\mathcal{Q}$ used in this definition is the set of quorums, and $\mathcal{B}$ is the fail-prone system.

Property 3. **M-Consistency**, $k > 1$: $\forall Q_1, Q_2 \in \mathcal{Q} : \forall B_1, B_2 \in \mathcal{B} : (Q_1 \cap Q_2) \setminus B_1 \not\subseteq B_2$

The paper then shows that if all sets in $\mathcal{B}$ have the same size t, then M-Consistency implies $n > 4t$.

We show that M-Consistency is equivalent to 4–Intersection. Since a faulty process can stop sending messages, we can use $\mathcal{S}_\Pi$ as the set of quorums: waiting to receive messages from more than a survivor set could prove fruitless. A fail-prone set is the complement of a survivor set, and for any two sets X and Y, $(X \setminus Y) \equiv (X \cap \bar{Y})$, where $\bar{Y}$ is the complement of Y. Hence, we can rewrite M-Consistency as:

$$\forall Q_1, Q_2 \in \mathcal{S}_\Pi : \forall B_1, B_2 \in \mathcal{B} : (Q_1 \cap Q_2) \cap \bar{B_1} \not\subseteq B_2$$

Then, for any two sets X and Y, $(X \not\subseteq Y) \equiv (X \cap \bar{Y} \neq \emptyset)$, and so:

$$\forall Q_1, Q_2 \in \mathcal{S}_\Pi : \forall B_1, B_2 \in \mathcal{B} : Q_1 \cap Q_2 \cap \bar{B_1} \cap \bar{B_2} \neq \emptyset.$$

Since the complement of a fail-prone set is a survivor set, this can be more compactly written as:

$$\forall Q_1, Q_2, S_1, S_2 \in \mathcal{S}_\Pi : Q_1 \cap Q_2 \cap S_1 \cap S_2 \neq \emptyset.$$

which is 4–Intersection. Hence, another way to write the replication requirement stated in M-Consistency is 4–Intersection, or equivalently 4–Partition.

4 An example of fractional k

The results of the previous section are perhaps not surprising to those who have designed Consensus or quorum algorithms. For example, 2–Intersection states that the survivor sets are a coterie, and 3–Intersection states that the intersection of any two survivor sets contains a nonfaulty process. It takes some effort to show that 4–Intersection is equivalent to M–availability, and we expect that it will not be difficult to show that the $n > 5t$ requirement of Fast Byzantine Paxos [22] can be understood from 5–Intersection. We conjecture that it is possible to define classes of algorithms that, as for Consensus [12,14], are built on top of quorums of various strengths, and whose communication requirements are easily understood in terms of k–Intersection. Such algorithms developed for the threshold model should be easily translatable into our model of non-IID failures. Potential candidates are algorithms for the abstractions in [9].

Less well understood are algorithms that have fractional replication predicates. To further motive the utility of intersection properties, we consider a problem that we call *Weak Leader Election*. Given a synchronous system and assuming receive-omission failures (that is, a faulty process can crash or fail to receive messages), the replication predicate for this problem is $n > \lfloor 3t/2 \rfloor$. This lower bound is not new, but to the best of our knowledge, it has not been shown that the lower bound is tight. We show here that the bound is tight, which is a result of some theoretical value. Our primary reason for choosing this algorithm, however, is the insight we used from the intersection property to arrive at the solution.

Leader Election arises in the context of Primary-Backup protocols. Leader election is an important problem in this context because a new primary has to emerge every time the current primary is non-responsive, and there must be a single primary at a time. Systems that implement a Primary-Backup protocol often run on a local area network. This has two implications. First, if a network with high transfer rates connects the processes, then it becomes important to consider receive-omission failures as processes may fail to receive messages due to buffer overflows [2]. Second, it enables the assumption of a synchronous system, which is common for Primary-Backup protocols.

We first specify the problem. Our specification allows for faulty (but non-crashed) processes to become elected. Such a feature is necessary because it requires more replication to detect receive-omission failures [23], and the original lower bound proof allowed such behaviors. We then discuss the lower bound on process replication for this problem using our model of dependent failures. Finally, we provide an algorithm showing that the lower bound is actually tight in our model.

4.1 Weak Leader Election

Each process p_i has a local boolean variable $p_i.elected$ ($p_i.elected$ is false for a crashed process). We then describe Weak Leader Election with two safety and two liveness properties.

Safety: $\Box(|\{p_i \in \Pi : p_i.elected\}| < 2)$.
LE-Liveness: $\Box\Diamond(|\{p_i \in \Pi : p_i.elected\}| > 0)$.
FF-Stability: In a failure-free execution, only one process ever has *elected* set to true.
E-Stability: $\exists p_i \in \Pi : \Diamond\Box(\forall p_j \in \Pi : p_j.elected \Rightarrow (j = i))$.

These properties basically state that infinitely often some process elects itself (LE-Liveness), and no more than one process elects itself at any time (Safety). The third property states that, in a failure-free execution, a single process is ever elected. This property, however, does not rule out executions with failures in which two or more processes are elected infinitely often. We hence define E-Stability.

4.2 Lower bound on process replication

In [3], the following lower bound was shown. The proof was given in the context of showing a lower bound on replication for Primary-Backup protocols.

Claim. Weak Leader Election for receive-omission failures requires $n > \lfloor 3t/2 \rfloor$.

Proof. Assume that Weak Leader Election for receive-omission failures can be solved with $n = \lfloor 3t/2 \rfloor$. Partition the processes into three blocks A, B and C, where $|A| = |B| = \lfloor t/2 \rfloor$ and $|C| = \lceil t/2 \rceil$. Consider an execution ϕ_A in which the processes in B and C initially crash. From LE-Liveness and E-Stability, eventually a process in A will be elected infinitely often. Similarly, let ϕ_B be an execution in which the processes in A and C crash. From LE-Liveness and E-Stability, eventually a process in B will be elected infinitely often.

Finally, consider an execution ϕ in which the processes in A fail to receive all messages except those sent by processes in A, and the processes in B fail to receive all messages except those sent by processes in B. This execution is indistinguishable from ϕ_A to the processes in A and is indistinguishable from ϕ_B to the processes in B. Hence, there will eventually be two processes, one in A and one in B, elected infinitely often, violating either Safety or E-Stability.

To develop the algorithm, we first generalize the replication predicate for this problem using cores and survivor sets. From the lower bound proof, we consider any partition of the processes into three blocks. Then, one constructs three executions, where in each execution all of the processes in two of the three subsets are faulty. The conclusion of the proof is the following property for $k = 3$:

Property 4. $(k, k-1)$–**Partition**, $k > 1$, $|\Pi| > 2$: $\exists k' \in \{2, \ldots, \min(k, |\Pi|)\} : \forall \mathcal{A} \in \mathcal{P}_{k'}(\Pi) : \exists \mathcal{A}' \in \mathcal{G}_{k'-1}(\mathcal{A}) : \exists C \in \mathcal{C}_\Pi : C \subseteq \mathcal{A}'$

The equivalent intersection property is then:

Property 5. $(k, k-1)$–**Intersection**, $k > 1$, $|\Pi| > 2$, $|\mathcal{S}_\Pi| > 2$: $\exists k' \in \{2, \ldots, \min(k, |\Pi|)\} : \forall \mathcal{T} \in \mathcal{G}_{k'}(\mathcal{S}_\Pi) : \exists T \in \mathcal{G}_2(\mathcal{T}) : (\cap_{S \in T} S) \neq \emptyset$

Stated more simply, $(k, k-1)$–Intersection says that for any set of k' survivor sets, $k' \in \{2, \ldots, \min(k, |\Pi|)\}$, at least two of them have a non-empty intersection. $(k, k-1)$–Intersection and $(k, k-1)$–Partition generalize replication predicates in the threshold model of the form

$n > \lfloor kt/(k-1) \rfloor$. Thus, a profile that satisfies $(k+1,k)$–Intersection must also satisfy $(k,k-1)$–Intersection. To illustrate, a system profile satisfies $(3,2)$–Intersection if either it satisfies $(2,1)$–Intersection or for every three survivor sets, two intersect. Also, note that $(2,1)$–Intersection is 2–Intersection.

Consider now an example of a system that satisfies $(3,2)$–Intersection. It is based on a simple two-cluster system. A process can fail by crashing, and there is a threshold t on the number of crash failures that can occur in a cluster. A cluster can also suffer a total failure, which causes all of the processes in that cluster to crash. Such a total failure can result from the failure of a cluster resource such as a disk array or a power supply, or from an administrative fault. We assume that total failures are rare enough that the probability of both clusters suffering total failures is negligible. However, processes can crash in one cluster at the same time that the other cluster suffers a total failure. Assuming that each cluster has three processes and $t = 1$, we have the following system profile, where processes with identifier a_i belong to one cluster and processes with identifier b_i belong to the other cluster:

Example 2.

$$\begin{aligned}
&\Pi = \{p_{a_1}, p_{a_2}, p_{a_3}, p_{b_1}, p_{b_2}, p_{b_3}\}\\
&\mathcal{C}_\Pi = \{\{p_{i_1}, p_{i_2}, p_{i_3}, p_{i_4}\} : (i_1, i_2 \in \{a_1, a_2, a_3\}) \wedge (i_3, i_4 \in \{b_1, b_2, b_3\} \wedge i_1 \neq i_2 \wedge i_3 \neq i_4)\}\\
&\mathcal{S}_\Pi = \{\{p_{i_1}, p_{i_2}\} : ((i_1, i_2 \in \{a_1, a_2, a_3\}) \vee (i_1, i_2 \in \{b_1, b_2, b_3\})) \wedge i_1 \neq i_2\};
\end{aligned}$$

To see why this profile satisfies $(3,2)$–Intersection, we just have to observe that out of any three survivor sets, at least two intersect.

The equivalence of $(k,k-1)$–Partition and $(k,k-1)$–Intersection can be shown with a proof similar to the one of Theorem 1, and we include it here for completeness.

Theorem 2. $(k,k-1)$–*Partition* $\equiv$ $(k,k-1)$–*Intersection*

Proof. $\Rightarrow$: Proof by contrapositive. Consider a system profile $\langle \Pi, \mathcal{C}_\Pi, \mathcal{S}_\Pi \rangle$ such that, for every k', $k' \in \{2, \ldots, \min(k, |\Pi|)\}$, there is a subset $\mathcal{S} = \{S_1, S_2, \ldots, S_{k'}\} \subseteq \mathcal{S}_\Pi$ such that $S_i \cap S_j = \emptyset$, $i \neq j$. That is, $\bigcup_{P \in \mathcal{G}_2(\mathcal{S})} \bigcap P = \emptyset$. We then build a partition $\mathcal{A} = \{A_1, A_2, \ldots, A_{k'}\}$ as follows:

$$\begin{aligned}
A_1 &= \Pi \setminus (S_2 \cup S_3 \cup \ldots \cup S_{k'})\\
A_2 &= \Pi \setminus (S_1 \cup S_3 \cup \ldots \cup S_{k'} \cup A_1)\\
&\vdots\\
A_i &= \Pi \setminus (S_1 \cup S_2 \cup \ldots \cup S_{i-1} \cup S_{i+1} \cup \ldots \cup S_{k'} \cup A_1 \cup A_2 \ldots \cup A_{i-1})\\
&\vdots\\
A_{k'} &= \Pi \setminus (S_1 \cup S_2 \cup \ldots S_{k'-1} \cup A_1 \ldots \cup A_{k'-1})
\end{aligned}$$

It is clear that A_i, A_j are disjoint, $i \neq j$. We now have to show that: 1) $\bigcup \mathcal{A} = \Pi$; 2) For every subset $\mathcal{A}' = \{A_{i_1}, A_{i_2}, \ldots, A_{i_{k'-1}}\} \subset \mathcal{A}$, $\bigcup \mathcal{A}'$ does not contain a core. To show 1), let $\psi_i = \cup_{(S_j \in \mathcal{S} \setminus S_i)} S_j$, $i \in \{1, \ldots, k'\}$. We then have the following derivation:

$$\begin{aligned}
\bigcup \mathcal{A} &= (\Pi \setminus \psi_1) \cup (\Pi \setminus \psi_2 \cup A_1) \cup \ldots \cup (\Pi \setminus (\psi_{k'} \cup A_1 \cup A_2 \ldots \cup A_{k'-1})) && (1)\\
&= \Pi \setminus ((\psi_1 \cap (\psi_2 \cup A_1)) \cap \ldots \cap (\psi_{k'} \cup A_1 \cup A_2 \ldots \cup A_{k'-1})) && (2)\\
&= \Pi \setminus (\psi_1 \cap \psi_2 \cap \ldots \cap (\psi_{k'} \cup A_1 \cup A_2 \ldots \cup A_{k'-1})) && (3)\\
&\vdots\\
&= \Pi \setminus (\cap_i \psi_i) && (4)\\
&= \Pi && (5)
\end{aligned}$$

- Line 1 to Line 2 follows from the observation that for any subsets A, B of Π, we have that $(\Pi \setminus A) \cup (\Pi \setminus B) = \Pi \setminus (A \cap B)$;
- Line 2 to Line 3: the intersection between ψ_1 and A_1 has to be empty, since ψ_1 contains exactly the elements we removed from Π to form A_1.

– Line 3 to Line 4: by repeating inductively the process used to derive Line 3, we are able to remove every term A_i present in the equation.
– Line 4 to Line 5: Transforming from a conjunctive form to a disjunctive form, we have that $\bigcap_{P\in\mathcal{G}_{k'-1}(\mathcal{S})}\bigcup P = \bigcup_{P\in\mathcal{G}_2(\mathcal{S})}\bigcap P$. To see why this is true, note that for every pair $S_i, S_j \in \mathcal{S}$, $i \neq j$, and $P \in \mathcal{G}_{k'-1}(\mathcal{S})$, we have that $(S_i \in P) \vee (S_j \in P)$. Finally, we have that $\bigcup_{P\in\mathcal{G}_2(\mathcal{S})}(\cap_{S_i\in P} S_i) = \emptyset$ by assumption.

By the construction of the partition and from the assumption that for every $S_i, S_j \in \mathcal{S}$, $S_i \cap S_j = \emptyset$, we have that for every $i \in \{1, \ldots, k'\}$, there is $S_i \in \mathcal{S}$ such that $S_i \subseteq A_i$. From this, we conclude that for any $\mathcal{A}' = \{A_{i_1}, A_{i_2}, \ldots, A_{i_{k'-1}}\} \subset \mathcal{A}$, $\bigcup \mathcal{A}'$ does not contain elements from some survivor set, and consequently it does not contain a core.

$\Leftarrow$: Proof also by contrapositive. Suppose a system profile $\langle \Pi, \mathcal{C}_\Pi, \mathcal{S}_\Pi \rangle$ such that for every k', $k' \in \{2, \ldots, \min(k, |\Pi|)\}$, there is a partition $\mathcal{A} = \{A_1, A_2, \ldots, A_{k'}\}$ of Π in which no union of $k'-1$ blocks of $\mathcal{A}$ contains a core. If a subset of processes does not contain a core, then it contains no elements from some survivor set. The complement of such a set of processes consequently contains a survivor set. Because no union of $k'-1$ blocks in $\mathcal{A}$ contains a core, for every A_i there is an $S_i \in \mathcal{S}_\Pi$ such that $S_i \subseteq A_i$. Thus, for all $A_i, A_j \in \mathcal{A}$, $i \neq j$, we have by construction that $A_i \cap A_j = \emptyset$, and hence $S_i \cap S_j = \emptyset$. We conclude that no pair $S_i, S_j \in \{S_1, S_2, \ldots, S_{k'}\}$ is such that $S_i \cap S_j \neq \emptyset$.

4.3 A Weak Leader Election algorithm

We now develop a synchronous algorithm WLE for Weak Leader Election for receive-omission failures. For this algorithm, we assume a system profile $\langle \Pi, \mathcal{C}_\Pi, \mathcal{S}_\Pi \rangle$ that satisfies $(3, 2)$–Intersection. WLE is round based: in each round a process receives messages sent in the previous round and then send messages to all processes. We use $p_i.M(r)$ to denote the set of messages that p_i receives in round r, and $p_i.s(r)$ to denote the set of processes from which process p_i receives messages in round r.

We developed this algorithm by first observing what $(3, 2)$–Intersection means. Given three survivor sets, at least two of them intersect. Put another way, if two survivor sets S_1 and S_2 are disjoint, then any survivor set S_3 intersects $S_1 \cup S_2$. Since a core is a minimal set that intersects every survivor set, the above implies that $S_1 \cup S_2$ contains a core. Thus, given any two disjoint survivor sets, at least one of them contains a correct process.

Our algorithm uses as a building block a weak version of Uniform Consensus that we call *RO Consensus*. We call it RO Consensus because of its resemblance to Uniform Consensus. RO Consensus, however, is tailored to suit the requirements of WLE and therefore is fundamentally different.

In RO Consensus, each process p_i has an initial value $p_i.a \in V \cup \{\perp\}$, where V is the set of initial values, and a decision value $p_i.d\,[1 \ldots n]$, where $p_i.d$ is a list and $p_i.d[j] \in V \cup \{\perp\}$. We use $v \in p_i.d$ to denote that there is some $p_\ell \in \Pi$ such that $p_i.d[\ell] = v$. If a process p_i crashes, then we assume that its decision value $p_i.d$ is $\mathcal{N}$, where $\mathcal{N}$ stands for the n element list $[\perp, \ldots, \perp]$. To avoid repetition throughout the discussion of our algorithm, we say that a process p_i decides in an execution ϕ if $p_i.d \neq \mathcal{N}$.

As we describe later, we execute our algorithm for RO Consensus, called ROC, multiple times in electing a leader. We then have that processes may crash before starting an execution ϕ of ROC. Such processes consequently have initial value undefined in ϕ. We therefore use $\perp$ to denote the initial value of crashed processes. That is, if $p_i.a = \perp$, then p_i has crashed.

We also use the relation $x \subseteq y$ for x and y lists of n elements to denote that: $\forall i, 1 \leq i \leq n$: $(x[i] \neq \perp) \Rightarrow (x[i] = y[i])$.

The specification of RO Consensus is composed of four properties as follows:

Termination: Every process that does not crash eventually decides on some value;

Agreement If $p_i.d[\ell] \neq \perp$, $p_i, p_\ell \in \Pi$, then for every non-faulty p_c, $p_i.d[\ell] = p_c.d[\ell]$;

RO Uniformity: Let *vals* be the following set: $\{d : \exists p_i \in \Pi \ s.t. \ (p_i.d = d)\} \setminus \mathcal{N}$. Then:

$\bigwedge$ $1 \leq |vals| \leq 2$
$\bigwedge$ $\forall d, d\prime \in vals : d \subseteq d\prime \vee d\prime \subseteq d$

$\bigwedge$ $\forall d_f, d_c \in vals, d_f \subseteq d_c : \exists S_f, S_c \in \mathcal{S}_\Pi$:
$\wedge\ \forall p \in S_f : (p \text{ crashes}) \ \vee\ (p.d = d_f)$
$\wedge\ \forall p \in S_c : (p.d = d_c) \ \wedge\ (p \text{ is not faulty})$

That is, there can be no more than two non-$\mathcal{N}$ decision values, and if there are two then one is a subset of the other. Furthermore, if there are two different decision values, then these are the values that processes in two disjoint survivor sets decide upon, one for the processes of each survivor set.

Validity: $\bigwedge$ If $p_j \in \Pi$ does not crash, then for all non-faulty p_i, $p_i.d[j] = p_j.a$
$\bigwedge$ If $p_j \in \Pi$ does crash, then exists $v \in \{\perp, p_j.a\}$ such that for all non-faulty p_i, $p_i.d[j] = v$
$\bigwedge$ If there are survivor sets $S_c, S_f \in \mathcal{S}_\Pi$ and values $v_c, v_f \in V, v_c \neq v_f$, such that:
$\wedge\ \forall p \in S_f : p.a \in \{v_f, \perp\}$
$\wedge\ \forall p \in S_c : ((p.a = v_c) \ \wedge\ (p \text{ is not faulty}))$
$\wedge\ \exists p_i, p_\ell \in \Pi : p_i.d[\ell] = v_f$
then for all p_j that does not crash, $v_f \in p_j.d$

That is, if a process p_i is not faulty and $p_i.d[j] \neq \perp$, then the value of $p_i.d[j]$ must be $p_j.a$. The value of $p_i.d[j]$, however, can be $\perp$ only if p_j crashes. The third case exists because we use the decision values of an execution as the initial values for another execution. From RO Uniformity, there can be two different non-$\mathcal{N}$ values d_f and d_c. If this is the case, then there is a survivor set S_c containing only correct processes such that all processes in S_c decide upon d_c, and another survivor set S_f containing only faulty processes such that all the processes in S_f either crash or decide upon d_f. Let v_f be d_f and v_c be d_c. By the third case, if some process that decides includes v_f in its decision value, then every process that does not crash also includes v_f in its decision value.

Figure 2 shows an algorithm that implements RO Consensus. In each round r, a process p_i collects messages and updates its list of initial values $p_i.A$. Once it updates $p_i.A$, p_i sends a message containing $p_i.A$ to all processes. A process p_i also assigns $p_i.A$ to $p_i.A_p(r)$ once it updates $p_i.A$ at round r. This enables p_i to verify in round $r+2$ if a process p_j has received the message p_i sent in round r. As we describe below, p_i uses $p_i.A_p(r)$ to determine if it is faulty.

ROC is an adaptation of a classic round-based synchronous Consensus algorithm for crash failures. There are two main differences. First, it uses survivor sets rather than a threshold scheme. It does use a constant t to bound the number of rounds; t is the number of processes subtracted the size of the smallest survivor set. Second, it has each process verify if it has committed a receive-omission failure. There are two ways that a process can notice this.

1. Recall that $p_i.s(r)$ is the set of processes from which process p receives a message in round r. Let $p_i.s(0)$ be an initial value, which for now consider to be the set of all processes. Since processes that have not decided or crashed send messages to all processes, for all non-faulty p_i that receives messages in rounds r and $r+1$: $p_i.s(r+1) \subseteq p_i.s(r)$. If this does not hold, then p_i must have failed to receive some message.
2. Consider a message m that p_i receives from p_j in round $r > 1$. Unless it crashes or discovers that it is faulty, a process sends a message to all processes in each round except the last. Let m' be the message that p_i sent to p_j in round $r-2$. If m indicates that p_j has not received m' ($p_i.A_p(r-2) \not\subseteq m'.A$), then p_i knows that p_j is faulty. Let $p_i.sr(r)$ be the processes in $p_i.s(r)$ with all processes that p_i knows to be faulty removed. By definition, we know that there is some survivor set that contains only correct processes. If $p_i.sr(r)$ does not contain a survivor set, then there is some correct process from which p_i did not receive a message. Hence, p_i can conclude it has failed to receive a message.

Note that RO Consensus differs from the definition of Uniform Consensus in that faulty processes may decide upon different values, although these values are not arbitrary and must be such as described by the RO Uniformity property. In the algorithm by Parvèdy and Raynal, for example, every process that decides must decide upon the same value [24].

Algorithm ROC on input $p_i.a$, $p_i.Procs$
round 0:
$p_i.s(0) \leftarrow p_i.Procs$; $p_i.sr(0) \leftarrow p_i.s(0)$
$p_i.A\,[i] \leftarrow p_i.a$
for all $p_k \in \Pi$, $p_k \neq p_i : p_i.A\,[i] \leftarrow \perp$
$p_i.A_p(0) \leftarrow p_i.A$
send $p_i.A$ to all

round 1:
$p_i.sr(1) \leftarrow p_i.s(1)$
if $\vee$ $p_i.s(1) \not\subseteq p_i.s(0)$
$\vee$ $\nexists S \in \mathcal{S}_\Pi : S \subseteq p_i.sr(1)$
then decide $[\perp, \ldots, \perp]$
else for each message $m_j \in p_i.M(1)$, $p_k \in \Pi$:
if $(p_i.A\,[k] = \perp)$ $p_i.A\,[k] \leftarrow m_j.A\,[k]$
$p_i.A_p(1) \leftarrow p_i.A$
send $p_i.A$ to all

round r: $2 \leq r \leq t$:
$p_i.sr(r) \leftarrow p_i.s(r) \setminus \{p_j : \exists m \in p_i.M(r) : p_i.A_p(r-2) \not\subseteq m.A \wedge m.from = p_j\}$
if $\vee$ $p_i.s(r) \not\subseteq p_i.s(r-1)$
$\vee$ $\nexists S \in \mathcal{S}_\Pi : S \subseteq p_i.sr(r)$
then decide $[\perp, ..., \perp]$
else for each message $m \in p_i.M(r)$, $p_k \in \Pi$:
if $(p_i.A\,[k] = \perp)$ $p_i.A\,[k] \leftarrow m.A\,[k]$
$p_i.A_p(r) \leftarrow p_i.A$
send $p_i.A$ to all

round $t+1$:
$p_i.sr(t+1) \leftarrow p_i.s(t+1) \setminus \{p_j : \exists m \in p_i.M(t+1) : p_i.A_p(t-1) \not\subseteq m.A \wedge m.from = p_j\}$
if $\vee$ $p_i.s(t+1) \not\subseteq p_i.s(t)$
$\vee$ $\nexists S \in \mathcal{S}_\Pi : S \subseteq p_i.sr(t+1)$
then decide $[\perp, ..., \perp]$
else for each message $m \in p_i.M(t+1)$, $p_k \in \Pi$:
if $(p_i.A\,[k] = \perp)$ $p_i.A[k] \leftarrow m.A[k]$
$p_i.A_p(t+1) \leftarrow p_i.A$
decide $p_i.A$

Fig. 2: ROC - Algorithm run by process p_i.

Informally, ROC satisfies RO Uniformity because $(3,2)$–Intersection holds. To decide on a value other than $\mathcal{N}$, a process must receive in each round messages from a set of processes that contains a survivor set. $(3,2)$–Intersection implies a low enough replication that there can be a set S of non-crashed faulty processes that communicate only among themselves. But, there cannot be two such sets S and S': if S and S' do not intersect, then $(3,2)$–Intersection implies that their union contains a core, violating the assumption that all of the processes in S and S' are faulty.

A set S of faulty processes that communicate only among themselves will decide on a value d where $d[i] = \perp$ for $p_i \notin S$ and $d[i] = p_i.a$ for $p_i \in S$. In addition, a correct process will also decide $d[i] = p_i.a$ for $p_i \in S$. Of course, a non-crashed faulty process can read from different sets of processes in each round, but by using the two rules given above, such a process can determine that it is faulty. Hence, at worst some faulty processes will decide on a value d_f and the correct processes will decide on a value d_c such that $d_f \subseteq d_c$.

The algorithm in Figure 3 uses ROC to implement Weak Leader Election. Algorithm WLE proceeds in iterations of an infinite repeat loop, where each iteration consists of two phases. In Phase 1, processes use ROC to distribute their process identifiers. In Phase 2, they use ROC to distribute what they decided on in Phase 1.

Informally, this algorithm satisfies Safety because of the following: it is possible for a set of faulty processes S to decide on the smaller value d_f in Phase 1, but by the end of Phase 2 the correct processes will know this. By Validity and RO Uniformity, every process that finishes Phase 2 uses the same list d_f to determine whether it is the current leader or not. Having the processes decide based on the smaller list d_f forces the receive-omission faulty processes to elect the same process as the correct processes. Note though, as mentioned above, that in this case the correct processes know that the elected process is faulty (although the elected process does not know).

LE-Liveness is obtained by repeatedly running the algorithm without resorting to a failure detector (which would require higher replication).

If there are no faulty processes, each election will always elect the process with the lowest identifier, which implies FF-Stability. To guarantee that there is no alternating behavior in which two processes are leaders infinitely often, non-crashed processes move forward the set of processes they believe are not crashed or have not stopped. That is, the input $p_i.Procs$ in ROC takes the value $p_i.s(t+1)$ from the previous execution of ROC (Π if it is the first execution of ROC). This implies E-Stability.

A formal proof of WLE appears in [17]. As WLE relies on the correctness of ROC, we also present in [17] a formal specification of this algorithm using TLA+ [19]. We model-checked this specification using TLC for some small models.

Algorithm WLE
$P \leftarrow \Pi$
repeat {
p_i.elected $\leftarrow$ FALSE
Phase 1:
  Run ROC with
    $p_i.a \leftarrow i$; $p_i.Procs \leftarrow P$
  $P \leftarrow p_i.s(t+1)$
  if $(p_i.d = [\perp, \ldots, \perp])$ then stop
Phase 2:
  Run ROC with
    $p_i.a \leftarrow p_i.d$ from Phase 1; $p_i.Procs \leftarrow P$
  $P \leftarrow p_i.s(t+1)$
  if $(p_i.d = [\perp, \ldots, \perp])$ then stop
  let $x \in p_i.d$ be a value such that $p_i.d[x] \neq [\perp, \ldots, \perp]$
    and it has the least number of non-$\perp$ values
  if (p_i is the first index of x such that $x[i] \neq \perp$)
    then p_i.elected $\leftarrow$ TRUE
}

Fig. 3: WLE - Algorithm run by process p_i.

5 Conclusions

In this paper we generalized a common argument used in proofs of lower bounds on process replication. The argument is based on the threshold model: it makes the assumption that, given n processes, any subset of $\lceil nb/k \rceil$ processes can be faulty. Then, after deriving a contradiction, the proof concludes that $n > \lfloor kt/b \rfloor$. In our generalization of the proof for $b = 1$, we conclude that k–Partition holds: if one partitions the processes into k subsets, then at least one of the subsets contains a core. Thus, lower bounds for many protocols can be trivially generalized for when process failures are not IID.

We then gave an equivalent property, k–Intersection, that is often useful when designing a protocol that takes advantage of non-IID process failures. Finally, we considered a problem for which the lower bound has $b = 2$. The lower bound on process replication for Weak Leader Election in a synchronous system with receive-omission failures was known to be $n > \lfloor 3t/2 \rfloor$, but this bound was not known to be tight. We showed that this bound is tight by first determining the intersection property for this replication predicate ($(3, 2)$–Intersection, equivalent to $(3, 2)$–Partition) and using it to guide our development of a protocol.

We have also studied further replication predicates for dependent failures. First, we have studied other problems that require the $(k, k-1)$–Intersection property for values of k greater than 3. One set consists also of problems of leader election. Finally, we have studied in more detail other dependent-failure replication predicates. Some of them are particularly interesting because they do not have an equivalent $n > \lfloor kt/b \rfloor$ in the threshold model. Such predicates arise, for example, in problems from Grid computing. One of our goals in studying replication predicates is to establish a hierarchy on the predicates based on implication. The main challenge has been to find problems that motivate the definition of a particular predicate.

Acknowledgements

We would like to express our gratitude to Geoffrey M. Voelker for useful insights, Marcos Aguilera for helpful discussions, and to the anonymous reviewers for comments that helped improving considerably this paper. Support for this work was provided by AFOSR MURI Contract F49620-02-1-0233.

References

1. B. Altmann, M. Fitzi, and U. Maurer. Byzantine Agreement secure against general adversaries in the dual failure model. In *Proceedings of DISC*, volume 1693/1999 of *LNCS*, pages 123–139. Springer-Verlag, Sep 1999.
2. Y. Amir, D. Dolev, S. Kramer, and D. Malkhi. Transis: A communication sub-system for high availability. In *22nd International Symposium on Faul-Tolerant Computing*, pages 76–84, San Francisco, CA, 1992.
3. N. Budhiraja, K. Marzullo, F. Schneider, and S. Toueg. Optimal Primary-Backup protocols. In *Proceedings of the 6th WDAG*, pages 362–378, Nov 1992.
4. M. Castro and B. Liskov. Practical Byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, Nov 2002.
5. M. Castro, R. Rodrigues, and B. Liskov. BASE: Using abstraction to improve fault tolerance. *ACM Transactions on Computer Systems*, 21(3):236–269, Aug 2003.
6. T. Chandra, V. Hadzilacos, and S. Toueg. The weakest failure detector for solving Consensus. *Journal of the ACM*, 43(4):685–722, Jul 1996.
7. T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, March 1996.
8. F. Christian. Synchronous Atomic Broadcast for redundant broadcast channels. *Journal of Real-Time Systems*, 2:195–212, Sep 1990.
9. R. Friedman, A. Mostefaoui, and M. Raynal. Intersecting Sets: A basic abstraction for asynchronous agreement problems. Technical Report PI-1598, IRISA, Rennes, France, Jan 2004.
10. H. Garcia-Molina and D. Barbara. How to assign votes in a distributed system. *Journal of the ACM*, 32(4):841–860, Oct 1985.
11. J. Gray and D. Siewiorek. High-availability computer systems. *IEEE Computer*, (9):39–48, Sep 1991.
12. R. Guerraoui and M. Raynal. A generic framework for indulgent Consensus. In *Proceedings of 23rd IEEE ICDCS*, pages 88–95, 2003.
13. M. Hirt and U. Maurer. Complete characterization of adversaries tolerable in secure multi-party computation. In *ACM PODC*, pages 25–34, Santa Barbara, California, 1997.
14. F. Junqueira and K. Marzullo. Consensus for dependent process failures. Technical Report CS2003-0737, UCSD, La Jolla, CA, Sep 2002.
15. F. Junqueira and K. Marzullo. Designing algorithms for dependent process failures. In *Proceedings of FuDiCo*, volume 2584/2003 of *LNCS*, pages 24–28. Springer-Verlag, Jan 2003.
16. F. Junqueira and K. Marzullo. Synchronous Consensus for dependent process failures. In *Proceedings of the 23rd IEEE ICDCS*, pages 274–283, May 2003.
17. F. Junqueira and K. Marzullo. Weak Leader Election in the receive-omission failure model. Technical Report CS2005-0829, UCSD, La Jolla, CA, Jun 2005.
18. K. Kursawe and F. Freiling. Byzantine fault tolerance on general hybrid adversary structures. Technical Report AIB-2005-09, Aachen University, Aachen, Germany, Jan 2005.
19. L. Lamport. *Specifying systems: The TLA+ language and tools for hardware and software engineers.* Pearson Education, Inc., 2002.
20. L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, Jul 1982.
21. D. Malkhi and M. Reiter. Byzantine Quorum Systems. In *29th ACM Symposium on Theory of Computing*, pages 569–578, May 1997.
22. J.-P. Martin and L. Alvisi. Fast Byzantine Consensus. In *Proceedings of DSN*, Jun 2005.
23. S. Mullender, editor. *Distributed Systems*, chapter 8. Addison-Wesley, 2nd edition, 1995.
24. P. R. Parvèdy and M. Raynal. Optimal early stopping Uniform Consensus in synchronous systems with process omission failures. In *Proceedings of the 16th ACM Symposium on Parallel Algorithms and Architectures*, pages 302–310, Barcelona, Spain, 2004.
25. F. B. Schneider. Implementing fault-tolerant services using the State-Machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, Dec 1990.