Title
Coping with Dependent Process Failures

Authors
Junqueira, Flavio
Marzullo, Keith
Voelker, Geoffrey M

Publication Date
2002-10-07

Peer reviewed

Coping with Dependent Process Failures *

Flavio P. Junqueira
flavio@cs.ucsd.edu

Keith Marzullo
marzullo@cs.ucsd.edu

Geoffrey M. Voelker
voelker@cs.ucsd.edu

Abstract

When developing fault-tolerant protocols, systems are usually modeled assuming that process failures are independent and identically distributed. In this paper, we present a system model that can represent correlated failures. We show that such a model is useful in that protocols can be made more efficient. Central to our approach is the idea of a core*, which is a reliable minimal subset of processes. We present two probabilistic failure models for dependent failures and discuss them in terms of computing cores. For both, finding a smallest minimal core is NP-hard, but one of the two models can be strengthened in a natural way to make computing a minimal core in P.*

Keywords: Reliability, Distributed Systems, Fault Tolerance, Dependent Failures, Computational Complexity
Submission category: regular paper
Word count: 10,500 **excluding optional appendices**

The material included in this paper has been cleared through authors' affiliations

Contact author: Flavio Junqueira
University of California, San Diego
9500 Gilman Dr., Dept. 0114
La Jolla, CA 92093-0114
USA

Phone number: (+1) 858 534-9669
CARTER Award: yes
Nominated student: Junqueira, Flavio
Advisor(s): Prof. Marzullo, Keith
Prof. Voelker, Geoffrey

*This work was developed in the context of the RAMP project, supported by DARPA as project number N66001-01-1-8933.

1 Introduction

Most fault-tolerant protocols are designed assuming that out of n components, no more than t can be faulty. For example, solutions to the Consensus problem are usually developed assuming no more than t of the n processes are faulty where "being faulty" is specialized by a failure model. We call this the *t of n assumption*. It is a convenient assumption to make. For example, bounds are easily expressed as a function of t: if processes can fail only by crashing, then the Consensus problem is solvable when $t < n$ if the system is synchronous and when $t < 2n$ if the system is asynchronous [1, 2].

The use of the t of n assumption dates back to the earliest work on fault-tolerant computing [3]. It was first applied to distributed coordination protocols in the SIFT project [4] which designed a fly-by-wire system. The reliability of systems like this is a vital concern, and using the t of n assumption allows one to represent the probabilities of failure in a simple manner. For example, if each process has a probability p of being faulty, and processes fail independently, then the probability $P(t)$ of no more than t out of n processes being faulty is:

$$P(t) = \sum_{i=0}^{t} \binom{n}{i} p^i (1-p)^{n-i}$$

If one has a target reliability R then one can choose for t the smallest value that satisfies $P(t) \geq R$.

The t of n assumption is best suited for components that have identical probabilities of failure and that fail independently. For embedded systems built using rigorous software development this can be a reasonable assumption, but for most modern distributed systems it is not. Processes' failures can be correlated because, for example, they were built with the same software that contains bugs.

That failures can have different probabilities and can be dependent is not a novel observation. The continued popularity of the t of n assumption, however, implies that it is an observation that is being overlooked by protocol designers. One reason that it is being overlooked is that one can use the t of n assumption for correlated failures; one just has to find a large enough value of t when deploying a system composed of a set of protocols. But, if one does so then the the resulting system can have superfluous redundancy. It would be better if one could take into account dependent failure information when designing a protocol.

This paper is a step towards allowing one to do so. In Section 2 we propose an abstraction that exposes dependent failure information for one to take advantage of in the design of a protocol. Like the t of n assumption, it is expressed in a way that hides its underlying probabilistic nature in

order to make it more generally applicable.

The abstraction is based on what we call a *core*, which is a minimal set of processes that do not all fail during any run of the protocol. We show in Section 3 that using this abstraction does not change the set of problems that are solvable, but it does allow for more efficient implementations. We then show in Section 4 that finding a smallest core is, in general, NP-hard and discuss some natural strengthenings of dependent failure assumptions that make finding a smallest core tractable.

There has been some work in providing abstractions more expressive than the t of n assumption. The hybrid failure model (for example, [5]) generalizes the t of n assumption by providing a separate t for different classes of failures. Using a hybrid failure model allows one to design more efficient protocols by having sufficient replication for masking each failure class. It is still based on failure in each class being independent and identically distributed. Quorum update has been developed that does take nonuniform behavior into account (for example, [6]) but an abstraction like the t of n assumption hasn't come out of this work.

2 System Model

In this section we describe an abstraction for dependent failures.

The following is a typical failure model that one finds in papers on fault-tolerant distributed protocols:

> We assume that there is a set of processes Π of size n. Each pair of processes is connected by a bidirectional communication channel. The channels are fair, in that they can drop messages, but if p repeatedly sends m to q and p and q do not fail, then q will eventually receive m. Channels do not spontaneously generate messages: if q receives m from p, then p had sent m, and q does not receive more copies of m than p sent.
>
> We assume that processes fail independently and no more than t of the n processes can fail. A process fails by crashing, where a crashed process takes no further steps.

The last paragraph is often not realistic. For example, if a process fails due to a software bug, then another process running the same software might also fail should it reach the same state or one similar enough to activate the bug. Another example is related to exploits in computer networks: the Code Red worm [7] affected hundreds of thousands of machines around the world that had in common the web server they were running.

The purpose of this paper is to start exploring the space of designing protocols for dependent process failures. Instead of assuming that processes fail independently, process failures are allowed to be correlated. With the assumption of dependent failures, the probability of a process failing during a run of a system depends on the history of failures in the run.

The t of n assumption is implicitly a probabilistic one: one chooses a value of t such that the probability of more than t processes failing is negligible. It is convenient for protocol designers to represent the target reliability in such a manner. We wish to have a similar convenient representation for the dependent failure assumption. We base ours on the notion of a *core*:

Definition 1 A core S is a minimal subset of Π such that, in any run of the system, at least one process in S does not crash.

The implicit assumption we make is that the probability of all the processes in a core failing is negligible (that is, it is less than $1 - R$ for some target reliability R).

A *cores set* is a set of all the cores of Π. A protocol designer can assume that in any run of the system, for each core S in the cores set, there will always be at least one process in S that does not crash. For example, consider a six process system $\Pi = \{p_1, p_2, p_3, p_4, p_5, p_6\}$. One could have a cores set $\{\{p_1, p_2, p_3\}, \{p_1, p_2, p_4\}, \{p_1, p_2, p_5\}, \{p_1, p_2, p_6\}\}$. This could result, for example, if p_1 and p_2 are jointly fairly reliable, but not reliable enough by themselves. Adding one more process results in a core. The other processes, however, have correlated failures and so by themselves do not provide sufficient reliability to be a core.

An equivalent way of representing dependent failures is with a *survivor set*. This representation is most simply explained using a little propositional logic. The cores set can be expressed as a proposition in conjunctive normal form, where the name of a process p is interpreted as a predicate meaning "p does not fail". For example, the cores set given above can be expressed as the conjunctive normal form proposition

$$(p_1 \vee p_2 \vee p_3) \wedge (p_1 \vee p_2 \vee p_4) \wedge (p_1 \vee p_2 \vee p_5) \wedge (p_1 \vee p_2 \vee p_6).$$

Rewritten into disjunctive normal form, the proposition is

$$p_1 \vee p_2 \vee (p_3 \wedge p_4 \wedge p_5 \wedge p_6).$$

The set of terms of such an expression—here, $\{\{p_1\}, \{p_2\}, \{p_3, p_4, p_5, p_6\}\}$—is the survivor set. In any run of the system, all of the processes in at least one of the elements of the survivor set will not fail.

The number of elements in the cores set and in the survivor set can be large. From a practical point of view, it is worthwhile to avoid explicitly using the entire cores set or the survivor set when possible.

Unlike with process failures, we do not introduce a novel model of channel failures. Thus, channels are assumed to drop messages, but not to permanently partition two processes. To cope with message losses, we assume that the two primitives **send**(m) and **receive**(m) are provided. These two primitives are assumed to implement a retransmission mechanism responsible for guaranteeing the delivery of a given message m. Since we assume fair channels, the channels become reliable by using these two primitives. In addition, these primitives satisfy the following two properties (commonly called *Validity* and *Uniform Integrity*):

- If p sends m to q, and both p and q are correct, then q eventually receives m;
- For any message m, q receives m at most once from p, and only if p previously sent m to q.

In terms of timing assumptions, the two extremes considered in the literature are the synchronous and asynchronous models. In the synchronous model, message delays, clock drift, and process speed are all bounded, whereas in the asynchronous model, no assumption is made about any of these system characteristics. Usually, real systems are neither totally synchronous nor totally asynchronous. They exhibit a mixed behavior, alternating between periods of asynchrony and synchrony. Several partially synchronous models try to capture this characteristic, such as the ones described in [8, 9, 10].

In Section 3, we show that modifying the system model to assume dependent failures does not change the set of solvable problems. It does, however, improve performance of protocols. We prove this claim by showing the tight bound on the number of steps required for consensus in the synchronous crash model with dependent failures.

3 Improving protocol performance by assuming dependent failures

In this section, we show that a model assuming dependent process failures allows for more efficient protocol implementations than a model assuming independent process failures. To support this claim, we first argue that a model assuming dependent failures is equivalent to the same model modified by assuming independent failures in terms of the problems that can be solved in both models. This result is not coupled to timing assumptions. We then illustrate the value of using a dependent failure model by proving a lower bound on the number of steps required for consensus

in the synchronous crash model with dependent process failures. Finally, we show that this bound on the number of steps is tight by giving a distributed algorithm that solves consensus in this number of steps.

3.1 Independent vs. dependent failure assumptions

The assumption that process failures may be correlated does not restrict or increase the set of problems that can be solved by the same model when assuming independent and identically distributed failures. To see this, assume a model M with some timing assumptions (synchronous, asynchronous, or partially synchronous). We call M_d the model M assuming dependent failures and M_i the model M assuming independent failures. We want to show that M_d and M_i are equivalent in terms of the set of problems that can be solved in these models. We show that one model can emulate the other.

It is easy to see that M_d can emulate M_i for any reasonable M_d. M_d should model a reasonable set of ways processes can fail. One reasonable way is when the processes fail independently and have identical probabilities of failure. To illustrate this, one dependent failure model that we discuss later in this paper is the *process-oriented* model. This model specifies a set of processes Π, and target reliability R and a function $Pf(p \in \Pi, S \subseteq \Pi)$ which is the probability that p has not failed given that all of the processes in S have failed. Modeling the t of n assumption is then:

- the set of processes Π remains the same;
- the mapping *Pf* is defined as:

$$Pf(p \in \Pi, S \subseteq \Pi) \begin{cases} x \text{ if } p \notin S \\ 1 \text{ otherwise} \end{cases}$$

- $R \leftarrow 1 - (1 - x)^t$

where *x* is a rational value $0 \leq x \leq 1$.

To show that M_i can emulate M_d we need to provide a value for t, the upper bound on the number of faulty processes in any execution of the system. The value of t has to be large enough so that any subset of processes that fail in an execution has a size no larger than t. We can find the smallest t from the survivor set *survivor*. The size of a smallest element $e \in survivor$ corresponds to the minimum number of processes that do not crash over all possible executions of the system. Therefore, if we make the value of t equal to $n - |e|$, then we guarantee that no more than t processes fail in any execution of the system.

3.2 Number of Steps for Synchronous Consensus

One important performance metric is the number of steps needed to solve consensus. In this problem, each process $p \in \Pi$ has a initial value $v_p \in \{0, 1\}$. The goal is to have all correct processes deciding on the same value v, which is proposed by some process either faulty or correct. Consensus satisfies the following four properties:

Validity : if a correct process p_i decides dec_i, then there is a process p_j such that $v_j = dec_i$

Integrity : every correct process decides on at most one value, and if it decides $v \neq \perp$ then some process proposed v;

Agreement : if some correct process p_i decides dec_i and another correct process p_j decides dec_j, then $dec_i = dec_j$;

Termination : every correct process eventually decides on some value.

Under the t of n assumption, it is well known that $t + 1$ steps are required to solve consensus[1]. Informally, this lower bound arises because, if the set of processes cannot yet decide on a consensus value, then the failure of a process can leave the remaining set of processes still undecided. If one structures the consensus protocol to first reach consensus within a smallest core S_{min}, then the maximum number of steps is at least $|S_{min}| - 1$.

In Appendix A we reproduce the lower bound proof from [11] on the number of steps to solve consensus in a synchronous system, rewritten to generalize the proof to apply to our dependent failure assumption. We show that this lower bound can be met with the consensus protocol of Figure 1. The protocol itself resembles the t of n consensus protocol in [12]. The original protocol presented a consensus service for the construction of failure-tolerant agreement protocols.

Claim 2 *DC solves consensus in a synchronous system.*

Proof: We need to prove that DC satisfy the four properties in the specification for consensus.

From the protocol, all nonfailed processes decide in the last round on the first value v different from $\perp$ found in the array of proposed values v_p. Because we initialize the array with $\perp$ in every position of v_p, the value of a position $v_p[i]$ only changes to $v \in \{0, 1\}$ if process i proposes v. Thus, DC satisfies Validity.

[1] Early stopping protocols can solve consensus using $\min(t+1, t'+2)$ steps where t' is the actual number of failures in the run. The same techniques can be used with the dependent failure assumption, and a tighter bound can be obtained. Due to space constraints we use the simpler case of non-early stopping protocols for this paper.

Algorithm DC **for process** p:
Initialization:

S is a core subset of processes
$v_p[1 \cdots |\Pi|]$ is an array of proposed values
For $i = 1$ to $|\Pi|$ do $v_p[i] = \perp$
$v_p[p] \stackrel{R}{\leftarrow} \{0, 1\}$

Round 0, $\forall p \in S$:

$m \leftarrow (p, v_p)$
send(m) **to** all process in S

Round $1 \leq t < |S| - 1$, $\forall p \in S$:

upon receive$(m = (q, v_q))$ **do**
for $i = 1$ to $|\Pi|$ **do**
if $(v_q[i] \neq \perp)$ **then** $v_p[i] \leftarrow v_q[i]$
$m' \leftarrow (p, v_p)$
send(m') **to** all processes in S

Round $|S| - 1$, $\forall p \in S$:

upon receive$(m = (q, v_q))$ **do**
for $i = 1$ to $|\Pi|$ **do**
if $(v_q[i] \neq \perp)$ **then** $v_p[i] \leftarrow v_q[i]$
$m' \leftarrow (p, v_p)$
send(m') **to** all processes in Π

Round $|S|$, $\forall p \in \Pi$:

upon receive$(m = (q, v_q))$ **do**
for $i = 1$ to $|\Pi|$ **do**
if $(v_q[i] \neq \perp)$ **then** $v_p[i] \leftarrow v_q[i]$
$i \leftarrow 0$
while $(v_p[i] = \perp)$ **do** $i \leftarrow i + 1$
$dec_p = (v_p[i])$

Figure 1: Synchronous Consensus for Dependent Failures

If a process p receives a value $v_{p'}$ in round $i : 1 \leq i < |S|$ and if p does not fail by the next round, then all processes q will have $v_q[p']$ set to $v_{p'}$. A simple inductive argument shows that if a nonfaulty process sets $v_q[p']$ in the last round, then it was sent in the previous round by a nonfault process. Consequently, the values that the nonfailed processes decide upon are the same, and so Agreement is satisfied.

Since Validity is satisfied, the value a process decides upon has to be proposed by some other process. In addition, by the construction of the protocol, every correct protocol decides on exactly one value. Hence, Integrity is satisfied.

Finally, Termination is trivially satisfied. Every execution of the protocol takes exactly $|S|$ steps.

□

To illustrate the benefits of assuming dependent failures, consider again the example presented in Section 2 of the six processes $\{p_1, p_2, p_3, p_4, p_5, p_6\}$ with cores set $\{\{p_1, p_2, p_3\}, \{p_1, p_2, p_4\}, \{p_1, p_2, p_5\}, \{p_1, p_2, p_6\}\}$ (and consequently the survivor set $\{\{p_1\}, \{p_2\}, \{p_3, p_4, p_5, p_6\}\}$). Any protocol for synchronous consensus assuming independent crash failures can be used in this system, but t would need to be five since the smallest element in the survivor set has a size of one. The number of steps required by this protocol is consequently $t + 1 = 6$. Our algorithm *DC*, however, only requires three steps, which is the size of the smallest core.

4 Probabilistic models of dependent failures and cores

Up to now we have argued that using the cores set abstraction can help one design more efficient fault-tolerant protocols, but we have not yet shown that the abstraction is sound. In this section we ask the question of whether the cores set abstraction can be supported by reasonable probabilistic models of dependent failures.

There are obvious problems of tractability associated with cores sets: the number of elements can be exponential in the number of processes. But, as we have seen in the previous section, in some cases we only need to know the identity of a single core. If there is some function $r(S)$ that computes in polynomial time the probability that at least one process in the set $S \subset \Pi$ of processes has not failed, then there is a trivial and efficient method of finding a core. If $r(\Pi) < R$ for a target reliability $R > 0$ then there is no core. Otherwise, initialize some variable C to Π. As long as there is a process $p \in C$ such that $r(C - \{p\}) \geq R$ then discard p. Since $r(\emptyset) = 0$, this loop will terminate, and the resulting value of C is a core.

The tractability of finding a *smallest* core depends on the probabilistic model one uses. To

illustrate the use and the consequences of choosing different probabilistic models, we describe two of them. One model associates probabilities of failure with sets of processes while the other associates probabilities of failure with attributes of processes. We call the first model the *process-oriented* model and the second the *attribute-oriented* model.

4.1 Process-oriented model

The elements of process-oriented model are as follows:

- A set of processes Π;
- A mapping $Pf : \Pi \times \mathcal{P}(\Pi) \rightarrow [0,1] \cap \mathbb{Q}$, where $\mathcal{P}(\Pi)$ is the power set of Π and $\mathbb{Q}$ is the set of rational numbers;
- A target degree of reliability R, where $0 < R \leq 1$

Pf evaluates to the probability that a process $p \in \Pi$ is correct given that all processes in a subset $S \subseteq \Pi$ are failed. More carefully, associate with every process an indicator variable that represents the current state of the process. We define the indicator variable X_p as follows:

$$X_p = \begin{cases} 0, & \text{if process } p \text{ has crashed} \\ 1, & \text{if process } p \text{ has not crashed} \end{cases}$$

Using our definition of indicator variables, we have the following interpretation of *Pf*:

$$\begin{array}{l} \text{For } p, q_1, q_2, \cdots, q_k \in \Pi \\ \quad Pf(p, \{q_1, q_2, \cdots, q_k\}) = \text{Pr}[X_p = 1 | X_{q_1} = 0, X_{q_2} = 0 \cdots, X_{q_k} = 0] \end{array}$$

There are two special cases that are worth mentioning. First, if the subset in the second parameter contains p, then the probability that p has not crashed is zero. Second, in the case that no process has failed, the second parameter is $\emptyset$ and the probability of the process in the first parameter being correct corresponds to the probability of it failing by itself. That is,

$$\begin{array}{rcl} Pf(p, \{p, q_1, \cdots, q_k\}) & = & \text{Pr}[X_p = 1 | X_p = 0, X_{q_1} = 0, X_{q_2} = 0 \cdots X_{q_k} = 0] = 0 \\ Pf(p, \emptyset) & = & \text{Pr}[X_p = 1] \end{array}$$

To define $r(S)$, we first set up a framework based on [13]. For a system with n processes, let $\mathbf{x} = (X_1, X_2, \cdots, X_n)$ be a vector of indicator variables representing the state of the system. Define the structure function $\phi(\mathbf{x})$ as follows:

$$\phi(\mathbf{x}) = \begin{cases} 0, & \text{if } \mathbf{x} \text{ denotes an invalid failure pattern} \\ 1, & \text{otherwise} \end{cases}$$

Let $\phi_S(\mathbf{x}_S)$ refers to the structure function associated with subset S, where $\mathbf{x}_S$ is a state vector representing the failure state of S:

$$\phi_S(\mathbf{x}_S) = 1 - \prod_{i \in S}(1 - X_i)$$

Function $\phi_S(\mathbf{x}_S)$ evaluates to 1 whenever the state vector $\mathbf{x}_S$ contains at least one indicator variable X_i equals to 1. Therefore, if there is at least one correct process in S, then we consider the subset S to be functioning as a core. We can now define $r(S)$:

$$\begin{aligned} r(S \subseteq \Pi) &= E[\phi_S(\mathbf{x}_S)] \\ &= Pr[\phi_S(\mathbf{x}_S) = 1] \end{aligned}$$

So, one computes $r(S)$ as follows:

$$\begin{aligned} & r(S = \{s_1, \cdots, s_k\}) = E[\phi_S(\mathbf{x}_S)] \\ &= E[1 - \prod_{i \in S}(1 - X_i)] \\ &= 1 - E[\prod_{i \in S}(1 - X_i)], \text{ by linearity of expectation} \\ &= 1 - (1 - Pr[\cup_{i \in S}(X_i = 1)]) \\ &= Pr[\cup_{i \in S}(X_i = 1)] \\ &= 1 - Pr[\cap_{i \in S}(X_i = 0)] \\ &= 1 - ((1 - Pf(s_k, \emptyset)) \cdot \prod_{i=0}^{k-1}(1 - Pf(s_i, \{s_{i+1}, s_{i+2}, \cdots, s_k\}))) \end{aligned}$$

If one wishes to represent *Pf* as a mapping, then an exponential space structure is required. To avoid this difficulty, we represent *Pf* as a time-bounded Turing machine, noting that in doing so we limit the situations for which this model can be applied.

We now argue that finding a smallest core is NP-hard. Define the decision problem associated with finding a smallest core SC as follows:

Input: A set of processes Π, a time-bounded Turing machine *Pf*, a target degree of reliability $R \in \mathbb{Q} \cap [0, 1]$, a positive integer k;

Question: Is there a subset $S \subseteq \Pi$ such that $r(S) \geq R$ and $|S| \leq k$?

To show that SC in NP , we need to provide a polynomial-time verifier that takes a certificate C and an instance $\langle \Pi, Pf, R, k \rangle$ and decides whether this instance is in SC or not. Unfortunately, we do not know how to verify in polynomial time whether the probability distribution associated to the Turing Machine is consistent or not. Hence, we don't know whether the input can be verified in polynomial time and hence don't know whether SC is in NP .

We show in Appendix B that SC is NP-hard by giving a reduction from the vertex cover problem. We provide here a brief description of this reduction. Assume an instance $\langle G = (V, E), k \rangle$ of the vertex cover problem [14]. Create a process for each vertex and associate the same probability of failure x with each edge. Edges fail independently, and a process fails when all the edges incident on that process fail. Consequently, the probability that a node p fails by itself is provided by the intersection of the events corresponding to the failure of the edges touching p. To build TM *Pf*, we need to define the probability that a process p has not failed given that some subset S has failed. If no neighbor of p is included in S, then $Pf(p, S)$ is equal to x to the power of the number of neighbors of p. Otherwise, $Pf(p, S)$ is equal to x to the power of the number of neighbors of p in S subtracted from the total number of neighbors of p. We set the target degree of reliability to be a function of x to the power of the total number of edges, which forces the solution to represent a vertex cover. Moreover, this vertex cover has size at most k if and only if the solution subset has also size at most k.

Since SC is NP-hard, the process-oriented model does not scale with the number of processes. If one has a large number of processes, then one would want to use approximation or randomized algorithms. Our work in this area so far leads us to conjecture that there is no polynomial time constant approximation algorithm.

4.2 Attribute-oriented model

Processes in real systems can be characterized by attributes, such as operating system, TCP/IP stack implementation, hardware version, etc. By identifying attributes that are critical to the correctness of the processes, it might be possible to infer failure dependencies. Processes are characterized by attributes that can fail. Once a process with a given attribute a_1 fails due to this attribute, we assume that all other processes with attribute a_1 will eventually fail. We assume that attributes fail independently.

We define attributes to be binary: for every possible attribute a, a process either has or doesn't have a. For instance, the attribute "OS_IS_LINUX" is owned only by those processes that run

Linux. One can think of other possible models in which attributes are not binary, but such a model can be translated to a binary attribute model. For example, suppose we have a model with n-ary attributes where all proceses have the same set of attributes. Each attribute a and each possible value x of a can be associated with a new attribute a_x. A process consequently either has or doesn't have a_x. By definition, a_x is a binary attribute.

We have in the attribute-oriented model a set of attributes Λ and two mappings f and *Pa*. f takes a process as a parameter and returns the set of attributes this process has. *Pa* evaluates to the probability of an attribute not failing. More formally, the model has the following components:

- a set Π of processes
- a set Λ of attributes
- a mapping $f : \Pi \rightarrow S_a \subseteq \Lambda$
- a mapping $Pa : \Lambda \rightarrow \mathbb{Q} \cap [0, 1]$
- a target degree of reliability $R \in [0, 1] \cap \mathbb{Q}$

With this model we define the structure function

$$\phi(\mathbf{x}) = 1 - \prod_{p \in S} (1 - \prod_{a \in f(p)} X_a)$$

where there is an indicator variable X_a for every attribute a. Then,

$$\begin{aligned} r(S) &= E[\phi(\mathbf{x})] \\ &= 1 - E[\prod_{p \in S} (1 - \prod_{a \in f(p)} X_a)] \end{aligned}$$

Finding a smallest core is also NP-hard in this model. The associated decision problem AS is:

Input: a set of processes Π, a set of attributes Λ, a mapping $f : \Pi \rightarrow \Lambda$, a mapping $Pa : \Lambda \rightarrow (0, 1) \cap \mathbb{Q}$, a target degree of reliability $R \in (0, 1) \cap \mathbb{Q}$, a positive integer $k \leq |\Pi|$;

Question: is there a subset $S \subseteq \Pi$ such that $r(S) \geq R$ and $|S| \leq k$?

We show that the AS problem is NP-hard in Appendix C by reduction from CLIQUE. Compared to SC, though, AS has the advantage of enabling the verification of the input in polynomial time. Because the probability of failure is expressed in terms of the attributes and the attributes fail independently, it is only necessary to verify that $Pa(a) \in (0, 1) \cap \mathbb{Q}$ for all $a \in \Lambda$.

The attribute-oriented model, however, can be strengthed to allow for polynomial computation of smallest cores. We give two such strengthenings next.

4.2.1 One Strengthening This first strengthening of the attribute-oriented model is plausible for a set of processes described by the same set of multi-valued attributes.

Assume a system with n processes and a fixed number of attributes a, uniformly distributed over the processes. Moreover, each process has a fixed number of attributes x and each attribute has the same probability π of failure. If we assume that $a >> x$, then we have several sets in which no pair of processes share an attribute. We say that a set of processes is a *disjoint set* if no two processes in the set share an attribute. The maximum number of process in an disjoint set is $\lfloor a/x \rfloor$ because there are at most $\lfloor a/x \rfloor$ disjoint combinations of attributes. The expected core size is the smallest i that satisfies $1 - (1 - \pi^x)^i \geq R$.

The following heuristic finds a smallest core satisfying R. The notation $p \stackrel{R}{\leftarrow} S$ means assign to p a value from S chosen randomly.

Algorithm *MVA* on input $\langle \Pi, A, f, p_a, R \rangle$
1- *Candidates* $\leftarrow \Pi$
2- $p \stackrel{R}{\leftarrow}$ *Candidates*
3- $S \leftarrow p$
4- while ($r(S) < R$ and *Candidates* $\neq \emptyset$) do
    4.1- Remove from *Candidates* all processes that share an attribute with p
    4.2- $p \stackrel{R}{\leftarrow}$ *Candidates*
    4.3- $S \leftarrow S \cup \{p\}$
5- output S

MVA first chooses a process p at random from Π and a set of candidate processes *Candidates* is set to Π. S, which will always be a disjoint set, is initialized to p. In the while loop starting on Step 4, all processes that share an attribute with p are discarded from *Candidates*; all the remaining processes share no attributes with the processes in S. One of the processes in *Candidates* is added to S and the loop continues as long as the desired reliability is met and *Candidates* is not empty. When the algorithm terminates, it outputs a disjoint set that may or may not be a core. (A user of this algorithm can check to see if it is a core by applying the function r to the output set.)

MVA is efficient in terms of time complexity. Steps 1, 2, and 3 take constant time. The while loop in Step 4 iterates at most n times, but each iteration does not take constant time. Removing processes that share attributes with the randomly chosen process p takes $O(x \cdot n^2)$ steps over all iterations. Therefore, *MVA* runs in $O(x \cdot n^2)$ steps.

Now we discuss how successful *MVA* is in finding a minimum core, assuming there is one. Let Z_k be random variable denoting the size of the set *Candidates* after k iterations of the while loop

Iterations	Number of processes		
	70	500	1000
1	50.43	360.22	720.44
2	34.89	249.26	498.52
3	22.93	163.79	327.58
4	14.06	100.49	200.98
5	7.84	56.03	112.06
6	3.79	27.09	54.18
7	1.44	10.34	20.68
8	0.34	2.46	4.92
9	0.01	0.12	0.24

Table 1: Expected size of *Candidates*

where $k \leq \lfloor a/x \rfloor$. Z_k has a binomial distribution:

- The number of trials is n, one for each process;
- A trial is done by checking if a randomly chosen process has attributes disjoint from a randomly chosen disjoint set of k processes. The probability of success $pr(k)$ is

$$pr(k) = \binom{a - k \cdot x}{x} / \binom{a}{x}.$$

Since Z_k has a binomial distribution, its expected value is $pr(k) \cdot n$.

Table 1 illustrates how the expected value of the size of *Candidates* decreases with the number of iterations. Note that $|S|$ is the iteration number plus one. It shows that even with a large number of processes the expected size of *Candidates* decreases with the number of iterations. This affects the chances of *MVA* returning a core. For example, if the expected size of a core is 10 (which is the size of S after 9 iterations), then for $n = 70$ the expected size of *Candidates* is small by nine iterations; the chances of *MVA* returning a core is low. If $n = 1000$, however, then the chances are much better.

As with any randomized heuristic, if we run the algorithm over the same input several times, then the probability of finding the desired output increases. To determine the minimum number of times we need to execute *MVA* to find a disjoint set of adequate size, we first compute the probability of finding such a disjoint set in one run of the algorithm. The probability that *Candidates* has at least one element is:

$$Pr[Z_k \geq 1] = 1 - (1 - pr(k))^n \text{ , for } k \leq \lfloor a/x \rfloor \quad (1)$$

As one would expect from Table 1, when the total number of processes increases, $(1 - pr(k))^n$ becomes smaller and the probability $Pr[Z_k \geq 1]$ increases. We can use Equation 1 to compute the probability that we find an independent set of appropriate size if we run the algorithm several times. In order to compute this probability, we define a second random variable X_k^r that corresponds to the number of times the algorithm finds a subset of size k when we run the algorithm r times. The following equation gives us the probability that X_k^r finds at least one such disjoint set by running *MVA* r times:

$$Pr[X_k^r \geq 1] = 1 - (1 - Pr[Z_k \geq 1])^r$$

Note that this probability is only meaningful if the total number of processes n in the system is much greater than $\lfloor a/x \rfloor$. Otherwise, it is simpler to run a deterministic algorithm that simply checks all possible combinations.

4.2.2 Second strengthening In the second strengthening of the attribute model, we associate with each process exactly one attribute (which we call its *color*). Formally, the model consists of:

- a set Π of processes
- a set C of colors
- a mapping $f : \Pi \to C$
- a mapping $Pc : C \to \mathbb{Q} \cap [0, 1]$
- a target degree of reliability $R \in \mathbb{Q} \cap [0, 1]$

The function $Pc(c)$ is the probability that color c does not fail during a run of the system. Because we assume that the failure of a color c_1 has no influence in the failure of another color c_2 and each process has exactly one color, we have that for $c_1 \neq c_2 : f^{-1}(c_1) \cap f^{-1}(c_2) = \emptyset$.

With this color model one can find a smallest core in polynomial time. The following algorithm illustrates this fact:

Algorithm *Color*: on input $\langle \Pi, C, f, Pc, R \rangle$
1- Sort colors by increasing order of failure probability
2- Let $Sorted[1 \cdots |C|]$ be the array of sorted colors
3- $S \leftarrow \emptyset$
4- $TR \leftarrow 1$ *probability of all in S failing*
5- for $i = 1$ to $|C|$
    5.1- $p \leftarrow$ any process in $f^{-1}(Sorted[i])$
    5.2- $S \leftarrow S \cup \{p\}$
    5.3- $TR \leftarrow TR \cdot (1 - Pc(C[i]))$
    5.4-if $(1 - TR \geq R)$ then output S
4- output S

The algorithm first sorts colors by increasing order of failure probability. It then traverses the array of sorted colors and stops when either it finds a core or reaches the end of the array. We need to show that the algorithm *Color* outputs a minimum core subset, whenever such a subset exists. Otherwise it outputs a minimal subset of processes with maximal reliability.

Claim 3 *Given an instance $I = \langle \Pi, C, f, \text{Pc}, R \rangle$ of the color model, algorithm* Color *outputs the smallest core subset for this instance if such a subset exists. Otherwise,* Color *outputs a smallest subset for which reliability is maximal.*

Proof: Given an instance $\langle \Pi, C, f, Pc, R \rangle$ of the color model, the maximum reliability achievable is $R_{\text{max}} = 1 - \prod_{c_i \in C}(1 - Pc[i])$. By the construction of the algorithm, if $R \leq R_{\text{max}}$, then it outputs a subset S such that $|S| \leq |C|$. Otherwise, *Color* returns a subset S of size exactly $|C|$.

It remains to be shown that the size of the core is minimum. There are two cases to consider. If there is at least one core in this instance, then assume that *Color* outputs a subset of processes S that is not the smallest. Thus, there is another core S' with $|S'| > |S|$. By the construction of the algorithm, if a process p is removed from S, then $r(S - \{p\}) < R$, which implies that $S' \not\subset S$. Because S' cannot be a subset of S, there is at least one process $p' \in S'$ such that $p' \notin S$. Moreover, p' is more reliable than at least one process in S. This is not possible, however, because the failure probabilities of all the processes $q \in S$ are smaller than or equal to the failure probabilities of the processes $p \in \Pi - S$. This is a contradiction, and so there is no such core S'.

If there is no core in the system, then $r(S) < R$ for every $S \in \mathcal{P}(\Pi)$. The maximum reliability achievable is $R_{\text{max}} = 1 - \prod_{c_i \in C}(1 - Pc[i])$ and a smallest subset that satisfies this condition contains exactly one process from each color. *Color* outputs one such subset in this case.

□

This algorithm is efficient in terms of the number of steps it takes to find a smallest core. Sorting the Colors in step 1 takes $\Theta(c \cdot log\ c)$ steps, where c is the number of colors. The loop in step 5 iterates at most C times and each step in an iteration takes $O(1)$ time. Notice that step 5.1 utilizes the inverse of f, which is constructed only once in time $\Theta(n)$, where n is the total number of processes. Therefore, the total running time of the algorithm is $\Theta(c \cdot \log c + c + n) = \Theta(c \cdot \log c + n)$.

4.3 Discussion

The tractability of finding a smallest core is not the only question one might have about a probabilistic model of dependent failures. The model has to be able to accurately represent the ways that processes fail. For example, our models restrict the ways that failures can be correlated. One

restriction they both impose is that the failure of a process p does not affect positively the probability of failure of a process q. In other words, no process becomes more reliable due to the failure of another process. The probability of failure of a process p either remains the same or increases with the failure of another process q. This doesn't seem to be a grave restriction: we are not aware of any real system for which this restriction does not apply.

A second restriction they impose is that failure probabilities and correlations do not change over time. As time advances in a run of the system, the failure probability of a process might change according to the failure of other processes, but the way it is affected is constant over time. For example, they do not model systems in which the failure of some process p makes the probability of failure of another process q increase for the next x minutes, nor do they model systems in which a process reliability increases over time as it "burns in". A third restriction is that failure probabilities are symmetric. For example, they do not model systems in which the failure of a process s makes the failure of c high, but the failure of c does not raise the probability of s failing.

The relation of probabilistic models of dependent failures with the cores set abstraction requires further study. An open question is whether the cores set abstraction is sufficiently general to model all reasonable probabilistic models. Our experience with the process-oriented and application-oriented models, though, indicate that the cores set abstraction has wide applicability.

5 Conclusions and Future Work

In this paper we have presented a new modeling abstraction for designing fault-tolerant algorithms when failures can be correlated. This abstraction, which is meant to serve in the same way that the t of n assumption served, is based on *cores*, which are sets of processes such that at least one of the processes will not fail in any execution. By using cores sets (or the equivalent abstraction of *survivor sets*) one can design protocols that can take advantage of the knowledge of how processes fail together. Doing so can result in more efficient protocols. We illustrated the benefit with a synchronous consensus protocol that can terminate faster.

This paper is one first step in the study of the impact of dependent failures on the design of fault-tolerant protocols. We are currently working on the following lines of research:

1. As noted in Section 4.3 our models of dependent failures are restricted. The details of the model of dependent failures determine the complexity (or, indeed, the computability) of core sets. Similarly, more restricted models of dependent failures can lead to more easily computed smallest cores. We are continuing to explore this question to better understand

the applicability of the cores set abstraction.

2. There are numerous protocols that have been designed using the t of n assumption. We are looking at a few of them—notably Consensus in synchronous and asynchronous systems and some information diffusion protocols such as [15]—to gain experience with designing protocols for dependent failures. Our hope is to develop a set of design rules for coping with dependent failures.

3. Determining failure probabilities, which underlie the t of n failure assumption, is difficult. Determining the correlated probabilities of failures, which underlie the abstraction of cores, is certainly no easier. In large open distributed systems it will most likely be impossible to know these probabilities with accuracy. We hope, though, that on-line techniques can be used to estimate these probabilities. With such techniques one could design a system to be adaptive as it determines how failures are correlated.

Acknowledgements We would like to thank our colleagues in the RAMP project (Stefan Savage, John Bellardo, Jessica Chiang, Sashka Davis, Marvin McNett, and Renata Teixeira), Andrè Barroso, Idit Keidar, Fred Schneider, and Dmitrii Zagorodnov for their support and comments on our work. We would also like to thank Mihir Bellare and Russell Impagliazzo for their guidance in the complexity results in this paper. Any errors in the results, however, are ours alone.

References

[1] T. Chandra, V. Hadzilacos, and S. Toueg, "The Weakest Failure Detector for Solving Consensus," *Journal of the ACM*, vol. 43, pp. 685–722, July 1996.

[2] T. Chandra and S. Toueg, "Unreliable Failure Detectors for Reliable Distributed Systems," *Journal of the ACM*, vol. 43, pp. 225–267, March 1996.

[3] J. von Neumann, "Probabilistic logics and the synthesis of reliable organisms from unreliable components," in *Automata Studies*, pp. 43–98, Princeton University Press, 1956.

[4] J. Wensley, "SIFT: Design and Analysis of a Fault-Tolerant Computer for Aircraft Control," in *Proceedings of the IEEE*, vol. 66, pp. 1240–1255, October 1978.

[5] F. Meyer and D. Pradhan, "Consensus with dual failure modes," in *Advances in Ultra-Dependable Distributed Systems* (N. Suri, C. J. Walter, and M. M. Hugue, eds.), IEEE Computer Society Press, 1995.

[6] D. K. Gifford, "Weighted voting for replicated data," in *Proceedings of the 7th ACM Symposium on Operating Systems Principles (SOSP)*, pp. 150–162, 1979.

[7] D. Moore, "The spread of the code-red worm (crv2)." http://www.caida.org/analysis/security/code-red/coderedv2_analysis.xml, July 2001.

[8] F. Cristian and C. Fetzer, "The Timed Asynchronous Distributed System Model," *IEEE Transactions on Parallel and Distributed Systems*, vol. 10, pp. 642–657, June 1999.

[9] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the Presence of Partial Synchrony," *Journal of the ACM*, vol. 35, pp. 288 – 323, April 1988.

[10] C. Delporte-Gallet and H. Fauconnier, "Synchronized Phase Systems," in *OPODIS 99*, (Hanoi), October 1999.

[11] I. Keidar and S. Rajsbaum, "On the Cost of Fault-Tolerant Consensus When There Are No Faults - A Tutorial," Tech. Rep. MIT-LCS-TR-821, MIT, May 2001.

[12] R. Guerraoui and A. Schiper, "Consensus service: A modular approach for building agreement protocols," in *Proceedings of the 26th International Symposium on Fault-Tolerant Computing*, pp. 168–177, June 1996.

[13] S. M. Ross, *Introduction to Probability Models*, ch. 9. Harcourt Academic Press, 7th ed., 2000.

[14] M. R. Garey and D. S. Johnson, *Computers and intractability: a guide to the theory of NP-completeness*. W. H. Greeman, 1979.

[15] M.-J. Lin, K. Marzullo, and S. Masini, "Gossip versus deterministic flooding: Low message overhead and high reliability for broadcasting on small networks," Tech. Rep. CS1999-0637, UCSD, 1999.

The reviewer need not read the following appendices to understand the paper. We include them here for any reviewer who wishes to examine the proofs we omitted from the main text because of space limitations.

A Lower bound for consensus in the synchronous crash model

To demonstrate the lower bound for the synchronous crash model with dependent failures, we use the technique of layering proposed by Keidar and Rajsbaum [11]. Beginning with a set of initial states, layers are applied consecutively to this set to show that the new set of states generated still contains states in which some correct process has not decided yet. A layer in this context is an action of the environment. This action can be, for example, messages dropped or processes failing.

More technically, layering is defined as a set of environment actions that can be performed by the system. The set of possible actions is coupled to the failure model assumed. In our case, we assume that a layering consists of crashing at most one arbitrary process at a given step. Moreover, before failing, a process can send messages to a number of process. Notice that a process p sends at most one message to another process q at each step. We then use $(p, [q])$ to denote that process p fails during this step, but the messages p sent to processes $\{1 \cdots q\} \subseteq \Pi$ in this step are received. Thus, we have the following layering for our model:

$$\mathbf{L} = \{(p, [q]) \mid p \in \Pi,\ [q] = \{1 \cdots q\} \subseteq \Pi\}$$

A layering is applied to a state. In this context, an initial state augmented by one or more applications of layering $\mathbf{L}$ is called an execution. We use $\mathbf{L}(x) = \{x \cdot l | l \in L\}$ to denote the application of layering $\mathbf{L}$ to state x and $L(X) = \{L(x) | x \in X\}$ to denote the application of layering $\mathbf{L}$ to the set of states X. In addition, we define $\mathbf{L}^i$ as the application of $\mathbf{L}$ for i consecutive times. This is expressed recursively as follows:

$$\begin{aligned} \mathbf{L}^0(X) &= X \\ \mathbf{L}^k(X) &= \mathbf{L}(\mathbf{L}^{k-1}(X)) \end{aligned}$$

We observe, however, that the number of consecutive layering applications is restricted to $|S| - 1$, where S is a minimum core. By assumption, at least one process in a core does not crash and after $|S| - 1$ applications not all layers $(i, [j]) \in \mathbf{L}$ can be applied. Consequently, the model restricts the number of consecutive applications of $\mathbf{L}$.

Another important definition is the one of similar states. Similarity of states captures the notion of states in which a correct process cannot make a decision, because there is not sufficient infor-

mation for it to do so. This notion is used extensively in the proofs presented below. Similar states and similarity connected sets of states are defined as follows:

Definition 4 States x and y are similar, denoted $x \sim y$, if there is a process p_j that is non-failed in these states, such that (a) x and y are identical except in the local state of p_j, and (b) there exists $p_i \neq p_j$ that is non-failed in both x and y. A set of states is similarity connected if for every $x, y \in X$ there are states $x = x_0, x_1, \cdots, x_m = y$ so that $x_i \sim x_{i+1}$, for all $0 \leq i \leq m$.

Each process p begins the execution of a consensus algorithm with a proposed value v_p. We assume a binary set of possible values. That is, we have that $v_p \in \{0, 1\}$. The set of initial states is consequently the set of all binary strings with length $|\Pi|$. We call this set $Init$ and we show that $Init$ is similarity connected with the following lemma.

Lemma 5 *$Init$ is similarity connected.*

Proof:

Given a state z, we denote by z_j the local state of process p_j in the state z. Let y, y' be two states in $Init$. For every $0 \leq m \leq n$, define x^m by setting $x^m_j = y_j$ for all $j > m$ and $x^m_j = y'_j$ for all $j \leq m$. We get: $x^0 = y$ and $x^n = y'$. Note that x^{m-1} and x^m differ exactly in the local state of process p_m. Since all the processes are non-failed in every state in $Init$, these states are similar, that is, $x^{m-1} \sim x^m$.

□

Now, we need to show that any k consecutive applications of layering $\mathbf{L}$ on a similarity connected set of states generates another similarity connected set of states. We note, however, that we can have no more than $|S|$ layering applications, by assumption. This is shown with the following lemma.

Lemma 6 *Let X be a similarity connected set of states in which no process is failed and S be a core, then $L^k(X)$ is similarity connected for all $k \leq |S|$.*

Proof: We prove by induction. The base case is $k = 0$. By definition, we have that $\mathbf{L}^0(X) = X$. Consequently, $\mathbf{L}^0(X)$ is similarity connected. The induction hypothesis is that $\mathbf{L}^{k-1}(X)$ is similarity connected and we want to show that $\mathbf{L}(\mathbf{L}^{k-1}(X))$ is also similarity connected. To show this, we need to demonstrate that the two following properties hold:

1. if $x \in \mathbf{L}^{k-1}(X)$ then $\mathbf{L}(x)$ is similarity connected;
2. if $y, y' \in \mathbf{L}^{k-1}(X)$, $y \sim y'$, then $\mathbf{L}(y) \cup \mathbf{L}(y')$ is similarity connected;

1: Suppose we apply layers $(i, [0])$ and $(j, [0])$ to x. Because no process is failed in none of these layers, we have that $x \cdot (i, [0])$ and $x \cdot (j, [0])$ are identical. Now let us apply layers $(i, [l-1])$ and $(i, [l])$ to x. $x \cdot (i, [l-1])$ and $x \cdot (i, [l])$ are either identical, in the case that process i did not send a message to l, or differ on the state of l, in which case they are similar.

2: y and y' differ in the state of one process, let's say i. If we apply layer $(i, [n])$ to both states, we get $y \cdot (i, [n])$ and $y' \cdot (i, [n])$. Notice that in this step, no process received a message from i. Moreover, all processes besides i have identical state in y and y' and consequently the messages they send have to be the same. Therefore, we have that $y \cdot (i, [n]) \sim y' \cdot (i, [n])$. Along with property 1, this proves our claim that $\mathbf{L}(y) \cup \mathbf{L}(y')$ is similarity connected.

□

We use the two previous lemmas to show a theorem that provides the lower bound on the number of steps required if $|S| - 1$ processes crash. The theorem is as follows:

Theorem 7 *Consider an algorithm for consensus in the synchronous crash model with dependent failures for a system with a core S. There exists an execution of the algorithm with* $|S| - 1$ *failures in which it takes at least* $|S|$ *steps for all correct processes to decide.*

Proof: By lemmas 5, the set of initial states is similarity connected. According to 6, the $|S| - 1$ applications of layering $\mathbf{L}$ on the set of initial states $Init$ results in another similarity connected set of states. Thus, there is some execution in which after $|S| - 1$ steps there is at least one correct process that has not decided yet. Therefore, at least $|S|$ steps are required for all correct processes to decide.

□

B SC is NP-hard

Claim 8 *Vertex Cover* $\leq_m$ *SC*

Proof:

We need to provide a polynomial-time algorithm A that returns an instance $\langle \Pi, Pc, R, sz \rangle$ of the SCproblem given an instance $\langle G = (V, E), k \rangle$ of the vertex cover (VC) problem, such that the following holds:

1. $\langle G = (V, E), k \rangle \in$ VC $\Rightarrow \langle \Pi, Pc, R, sz \rangle \in$ SC
2. $\langle \Pi, Pf, R, sz \rangle \in$ SC $\Rightarrow \langle G = (V, E), k \rangle \in$ VC

We describe the algorithm A as follows:

A on input $G = (V, E)$, k
1- $\Pi \leftarrow \emptyset$
2- $x \leftarrow 0.2$ (arbitrary choice)
3- For every vertex $u \in V$
3.1- Create a new process labeled u
3.2- $\Pi \leftarrow \Pi \cup \{u\}$
4- Make function *Pf* as follows:
Pf on input $p \in \Pi$, $S \subseteq \Pi$
if $p \in S$ then return 1
$nb \leftarrow$ number of neighbors of p
if $S = \emptyset$ then return x^{nb}
$S' \leftarrow S$
$i \leftarrow 0$
while $S' \neq \emptyset$
$q \stackrel{R}{\leftarrow} S'$
$S' \leftarrow S' - \{q\}$
if $(p, q) \in E$ then $i \leftarrow i + 1$
return x^{nb-i}

5- $R \leftarrow 1 - x^{|E|}$
6- $sz \leftarrow k$
7- Output $\langle \Pi, Pf, R, sz \rangle$

The algorithm clearly runs in polynomial time on the input $\langle G, k \rangle$. One detail about the algorithm that needs clarification is the arbitrary choice of probability x. In this reduction, we can see x as the probability of an edge $e \in E$ failing. A process $p \in \Pi$ is faulty if and only if all the edges touching p fail. Hence, assuming a sample space in which samples represent edges failing, some samples may not contain sufficient faulty edges to make any process faulty. Moreover, these samples have probability greater than or equal to 0 (zero). Suppose E_p corresponds to the event in which $p \in \Pi$ is faulty, we then have the following:

$$\begin{aligned} 0 \leq Pr[\cup_{p \in \Pi} E_p] &\leq \binom{|E|}{1} \cdot x - \binom{|E|}{2} \cdot x^2 + \cdots + (-1)^{k+1} \binom{|E|}{k} \cdot x^k + \\ &\quad + \cdots + (-1)^{|E|+1} \binom{|E|}{|E|} \cdot x^{|E|} \\ &= (-1)^{|E|-1}(x-1)^{|E|} + 1 \text{ , for } |E| \geq 1 \end{aligned}$$

We need to determine for which values of x Equation 2 is satisfied. For odd $|E|$, we have the following three constraints:

$$\begin{aligned} 0 < x < 1 & \\ (x-1)^{|E|} < 0 &\Rightarrow x < 1 \\ -1 < (x-1)^{|E|} &\Rightarrow x > 0 \end{aligned}$$

For even $|E|$, we have the following three constraints:

$$\begin{aligned} 0 < x < 1 & \\ -(x-1)^{|E|} < 0 & \Rightarrow \quad x \neq 1 \\ -1 < -(x-1)^{|E|} & \Rightarrow \quad 0 < x < 2 \end{aligned}$$

Therefore, any value in the range $(0, 1)$ is appropriate. It remains to show that properties 1 and 2 hold for A.

First, we prove 1. Let us assume that a graph $G = (V, E)$ has a subset V' of size at most k such that for every edge $(u, v) \in E$ either u or v is in V'. We can build a solution S for the instance $\langle \Pi, Pf, R, sz \rangle$ by including in S the processes associated with the vertices in V'. There are two cases to be analyzed. In the first one, no pair of nodes in V' are neighbors. More formally, for every $u, v \in V', u \neq v$, we have that $(u, v) \notin E$. According to the algorithm, for every process $p \in S$, we have that $Pf(p, S - \{p\}) = x^{nb(p)}$. Thus, if V' is a vertex cover for G and we apply 1, we have that the reliability of S is exactly $1 - x^{|E|}$. Now, let us assume that there is at least one pair of vertices $u, v \in V'$, such that $(u, v) \in V'$. In this case, according to the construction of *Pf* and equation 1, the reliability of S is exactly $1 - x^{|E|}$, because in computing the probability of all processes in S being faulty, we multiply x exactly once for every edge in E.

Now, we show 2. If there is a subset of processes S of size at most $sz = k$ that satisfies the target degree of reliability $R = 1 - x^{|E|}$, then a vertex cover V' for G is composed of the vertices associated with the processes in S. According to our algorithm and equation 1, for every edge $(u, v) \in E$, we multiply x exactly once in the computation of the probability of all processes being faulty, even if both the processes labeled u and v are in S. Therefore, to obtain reliability R, S has to be composed of processes associated to vertices such that for every edge $(u, v) \in E$ either process u or process v is in S. We conclude that V' is a vertex cover for G of size at most k.

□

C AS is NP-hard

We prove the claim that the problem AS is NP-hard by providing a reduction from CLIQUE as follows:

Claim 9 CLIQUE$\leq_m$ AS

Proof: We need to provide a polynomial-time A_{AS} algorithm that takes an instance $\langle G = (V, E), k \rangle$ of the the CLIQUE problem and output an instance $\langle \Pi, \Lambda, f, Pa, R, k' \rangle$ of the AS problem such that:

1. if $\langle G = (V, E), k\rangle \in$ CLIQUE then $\langle \Pi, \Lambda, f, Pa, R, k'\rangle \in$ AS

2. if $\langle G = (V, E), k\rangle \notin$ CLIQUE then $\langle \Pi, \Lambda, f, Pa, R, k'\rangle \notin$ AS

We describe A_{AS} as follows:

Algorithm A_{AS}
On input $\langle G = (V, E)\rangle$
1- $\Pi \leftarrow \emptyset$; $\Lambda \leftarrow \emptyset$
2- For every $u \in V$
    2.1- Create process p_u
    2.2- $\Pi \leftarrow \Pi \cup \{p_u\}$
3- For every $p_u \in \Pi$
    3.1- Create attribute a_u
    3.2- $f(p_u) \leftarrow \{a_u\}$
    3.3- $\Lambda \leftarrow \Lambda \cup \{a_u\}$
4- For every $u, v \in V, u \neq v$, such that $(u, v) \notin E$ do
    4.1- Create attribute $a_{u,v}$
    4.2- $f(p_u) \leftarrow f(p_u) \cup \{a_{u,v}\}$;$f(p_v) \leftarrow f(p_v) \cup \{a_{u,v}\}$
    4.3- $\Lambda \leftarrow \Lambda \cup \{a_u v\}$
5- $x \leftarrow 0.002$, %probability of process p crashing by itself
6- For every $u, v \in V, u \neq v$, such that $(u, v) \notin E$ do
    6.1- $r \leftarrow \max\{|f(p_u)| + 1, |f(p_v)| + 1\}$
    6.2- $Pa(a_{uv}) \leftarrow 1 - \sqrt[r+1]{x}$
7- For every $u \in V$ do
    7.1- $Pa(a_u) \leftarrow \frac{x}{(\prod_{a \in f(p_u), a \neq a_u}(1 - Pa(a)))}$
8- $k' \leftarrow k$
9- $R \leftarrow 1 - (1 - 0.002)^k$
10- Output $\langle \Pi, \Lambda, f, Pa, R, k'\rangle$

We argue that A_{AS} runs in polynomial time. The first five steps clearly run in polynomial time. Steps 1 and 5 take constant time, while 2, 3 and 4 iterate a polynomial number of times on the size of either the set of vertices or the set of edges. Each statement in these loops executes in polynomial time, since they are creating new elements, such as processes and attributes, and adding these elements to their respective sets. Steps 6 and 7 execute expensive operations such as n^{th} root and multiplication, but multiplication and rational approximation of n^{th} root can be done in polynomial time. Rational approximation introduces an error to the computation, but as we argue below this error does not impact the correctness of the reduction. Finally, computing the target degree of reliability in step 9 requires k multiplications and two subtractions, which turns out to be polynomial on the integer k.

The algorithm works by creating an attribute a_{uv} for every edge (u, v) in the complement graph G'. This attribute is shared by the processes touched by the edge. Every process p_u has also

an attribute a_u which is not shared with any other process. The main reason for having this extra attribute is to equalize the probability of failure of the processes. Any error that is introduced by the rational approximation of extracting the n^{th} root can be compensated for with these probabilities for the attributes a_u. Although the processes have different failure correlations, every process has the same probability of failing by itself. Intuitively, by doing this, we avoid having very reliable processes that are not present in a clique.

We now need to show that properties 1 and 2 hold for A_{AS}.

1: Assuming there is a clique of size k in G, there is a subset of processes $S \subseteq \Pi$ such that no pair of processes in S share an attribute and S has size k. Consequently, the processes in S fail independently and the reliability is exactly $1 - (1 - 0.002)^k$.

2: Assuming there is no clique of size k in G, there is no subset of processes $S \subseteq \Pi$ such that no pair of processes in S share an attribute and $|S| = k$. Because all the processes have the same probability of crashing and every subset of size k has correlated failures, there is no subset of size less than or equal to k that satisfies the target degree of reliability $1 - (1 - 0.002)^k$.

This concludes our proof.

□