Coterie availability in sites (extended version)

Flavio Junqueira and Keith Marzullo

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

Abstract. *In this paper, we explore new failure models for multi-site systems, which are systems characterized by a collection of sites spread across a wide area network, each site formed by a set of computing nodes running processes. In particular, we introduce two failure models that allow sites to fail, and we use them to derive coteries. We argue that these coteries have better availability than quorums formed by a majority of processes, which are known for having best availability when process failures are independent and identically distributed. To motivate introducing site failures explicitly into a failure model, we present availability data from a production multi-site system, showing that sites are frequently unavailable. We then discuss the implementability of our abstract models, showing possibilities for obtaining these models in practice. Finally, we present evaluation results from running an implementation of the Paxos algorithm on PlanetLab using different quorum constructions. The results show that our constructions have substantially better availability and response time compared to majority coteries.*

1 Introduction

There has been a proliferation of large distributed systems that support a diverse set of applications such as sensor nets, data grids, and large simulations. Such systems consist of multiple sites connected by a wide area network, where a site is a collection of computing nodes running one or more processes. The sites are often managed by different organizations, and the systems are large enough that site and process failures are common facts of life rather than rare events.

Critical services in such systems can be made highly available using replication. In data grids, for example, data sets are the most important assets, and having them available under failures of sites is very desirable. To improve availability, the well-known *quorum update* technique can be used. This technique consists of implementing a mutual exclusion mechanism by reading and writing to sets of processes that intersect (*quorums*) [8]. As another example, the Paxos protocol [18] enables the implementation of fault-tolerant state machines for asynchronous systems. Paxos is a popular choice because of its ability to produce results when a majority of replicas survive, for its feature of not producing erroneous results when failures of more than a majority (indeed, up to a complete failure) occur, and its very weak assumptions about the environment. Underlying Paxos (and other similar protocols) is the same quorum update technique.

This paper considers quorum constructions for multisite systems. The problem area of quorums for multisite systems is large and not well studied. We address a set of problems from this area as an early foray. We first give a failure model for multisite systems that is simple and has intuitive appeal, and then give a second failure model that has less intuitive appeal but theoretical and practical interest. Because sites can fail, the failures of processes are not independent, and so an IID (independent, identically distributed) model is not appropriate. We define a new metric for availability that is suitable to non-IID failures, and give optimal quorum constructions for both models. We discuss the implementability of the two failure models, and discuss an experiment of running Paxos on PlanetLab [23] that gives some validation of our results.

Related work Quorum systems have been studied for over two decades. The first algorithms based on quorums use voting [10]. Garcia-Molina and Barbara generalized the notion of voting mechanisms, and proposed the use of minimal collections of intersecting sets, or *coteries* [8]. Most of the following work (such as [17,20,22]) has concentrated on how quorums can be constructed to give good availability, load and capacity assuming relatively simple system properties (such as identical processes and independent failures) [2,3,21]. Only recently the problem of choosing quorums according to properties of the system (such as location) has attracted some attention [11,16]. Of particular interest to our current work are the constructions of [17] and [7]. In [17], Kumar proposed, to the best of our knowledge, the first hierarchical quorum construction, and showed that by doing so one can have smaller quorums. The analysis in [17], however, assumes IID failures. The work by Busca *et al.* assumes a multisite system similar to what we assume here, and their quorum construction [7] is very similar to our *Qsite* construction. Their focus, however, was on performance. If one considers the distribution of response times from a quorum system, performance is often measured using the average or median, while availability is a property of the tail of the distribution. Thus, high performance does not necessarily imply high availability. Availability in quorum systems has been studied before [2,3,21], but we argue here that the previous metrics are not suitable for multisite systems. A notable exception is the work by Amir and Wool [1], which evaluates several existing quorum constructions in the context of a small, real network.

A *network partition* is a failure event that leads to one set of nonfaulty processes being unable to communicate with another set of nonfaulty processes (and, often, vice versa). Quorum systems are asynchronous, and so a network partition is treated identically to slow-to-respond processes. Long-lasting network partitions can make it impossible to obtain a quorum. A recent paper by Yu presents a probabilistic construction that does increase availability in the face of partitions, but it assumes a uniform distribution of servers across the network [26]. In comparison, our constructions are deterministic and make no assumption about distributions of sites.

2 System model

We consider a system of a set P of processes. The processes are partitioned into *sites* $\mathcal{B} = \{B_1, B_2, \ldots, B_k\}$, and between each pair of processes there is a bidirectional

communication channel. Processes can fail by crashing, and a crashed process can recover. Similarly, a site can fail and recover. A site failure represents the loss of a key resource used by the processes in the site (such as network, power, or a storage server) or some event that causes physical damage to the equipment on the site (such as loss of A/C); the processes in the site are all effectively crashed while the site is faulty.

Let $\mathcal{E}$ represent the executions of the system. Each execution $E \in \mathcal{E}$ is a sequence of system states. Each state $s \in E$ of an execution has an associated *failure pattern* $F(s, E) \subseteq P$, which is the set of processes that are faulty in s. If site B_i is faulty in s, then all of the processes in B_i are in $F(s, E)$. We use $NF(E, s) = P \setminus F(E, s)$ to denote the set of nonfaulty processes in s. We say that a failure pattern f is valid iff $\exists E \in \mathcal{E} : \exists s \in E : f = F(E, s)$.

We use survivor sets to express valid failure patterns. Survivor sets were introduced in [14] to provide a more expressive model of process failures. Informally, a survivor set is a minimal subset of nonfaulty processes. There are different ways to define survivor sets more formally: we have used probabilities [14] and have used the complement of maximal failure patterns [15]. We use the second one here. This definition does not rely on probabilities directly, although failure probabilities can be used to determine survivor sets; we discuss this point later in this paper. The definition is:

Definition 1. *Given a set of processes P, a set S is a survivor set if and only if:*[1]

$$\begin{array}{l} \bigwedge S \subseteq P \\ \bigwedge \exists E \in \mathcal{E} : \exists s \in E : S = \text{NF}(E, s) \\ \bigwedge \forall p \in S : \forall E \in \mathcal{E} : \forall s \in E : S \setminus p \neq \text{NF}(E, s) \end{array}$$

We use $\mathcal{S}_P$ to denote the set of survivor sets of P, and we call a pair $\langle P, \mathcal{S}_P \rangle$ a *system profile*.

We now repeat a few definitions that have appeared elsewhere and that we use in this paper. A coterie $\mathcal{Q}$ is a set of subsets of P that satisfies the following two properties [8]: 1) $\forall Q_i, Q_j \in \mathcal{Q} : Q_i \cap Q_j \neq \emptyset$; 2) $\forall Q_i, Q_j \in \mathcal{Q}, Q_i \neq Q_j : Q_i \not\subset Q_j \wedge Q_j \not\subset Q_i$. The first property is called 2-Intersection [15], and it says that quorums in a coterie pairwise intersect. This property guarantees mutual exclusion when executing operations on quorums, such as reads and writes, as every pair of quorums must have at least one process in common. The second property states that all quorums are minimal. A coterie $\mathcal{Q}$ is *dominated* if there is a coterie $\mathcal{Q}'$ such that: 1) $\mathcal{Q} \neq \mathcal{Q}'$; 2) $\forall Q \in \mathcal{Q} : \exists Q' \in \mathcal{Q}' : Q' \subseteq Q$. If no coterie dominates a coterie $\mathcal{Q}$, then we say that $\mathcal{Q}$ in *non-dominated*.

A *transversal* of a coterie is a subset of processes that intersects every quorum in the coterie. We use $\mathcal{T}(\mathcal{Q})$ to denote the set of transversals of the coterie $\mathcal{Q}$. Transversals are useful for defining the availability of a coterie: a coterie $\mathcal{Q}$ is available in a step s of some execution E if and only if $F(s, E) \notin \mathcal{T}(\mathcal{Q})$.

[1] We use the "bulleted conjunction" and the "bulleted disjunction" notation list invented in TLA$^+$ [19]. In Definition 1, the list corresponds to the conjunction of the statements to the right of the "$\bigwedge$" marks.

3 Computing availability

The availability of coteries can be computed in various ways. One metric is *node vulnerability* which is the minimum number of nodes that, if removed, make it impossible to obtain a quorum [3]. A similar metric, *edge vulnerability*, counts the minimum number of channels whose removal makes it impossible to obtain a quorum (no connected component contains a quorum). Both of these metrics are appropriate when failures are independent and identically distributed (IID) because they measure the minimum number of failures necessary to halt the system. They are not necessarily good metrics for multisite systems. Consider the following threesite system in which a survivor set is the union of majorities of processes in a site for some majority of sites:[2]

$$\begin{aligned} P &= \{a_1, a_2, a_3, b_1, b_2, b_3, c_1, c_2, c_3\} \\ \mathcal{B} &= \{a_1a_2a_3, b_1b_2b_3, c_1c_2c_3\} \\ \mathcal{S}_P &= \{a_ia_jb_lb_m : i,j,l,m \in \{1,2,3\} \wedge\ i \neq j \wedge l \neq m\} \\ &\quad \cup \{a_ia_jc_lc_m : i,j,l,m \in \{1,2,3\} \wedge\ i \neq j \wedge l \neq m\} \\ &\quad \cup \{b_ib_jc_lc_m : i,j,l,m \in \{1,2,3\} \wedge\ \ i \neq j \wedge l \neq m\} \end{aligned}$$

From our system model, processes are pairwise connected. According to the results in [3], the best strategy for both node and edge vulnerability is then to use quorums formed of majorities, which for this system is any subset of five processes. By definition, for every $S \in \mathcal{S}_P$, there is some step s of some execution $E \in \mathcal{E}$ such that $S = F(s, E)$, where $\mathcal{E}$ is the set of executions of $\langle P, \mathcal{S}_P \rangle$. As P contains nine processes and every $S \in \mathcal{S}_P$ contains four processes, there are five faulty processes in such a step, and hence no majority quorum can be obtained. If one uses $\mathcal{S}_P$ as a coterie, however, then there is one quorum available in every step, by construction. $\mathcal{S}_P$ has therefore better availability than the majority construction.

An alternative to node and edge vulnerability is, given probabilities of failures, to directly compute the probability of the most likely failure patterns that make it impossible to obtain a quorum. Probability models, however, can become quite complex when failures are not IID. To avoid such complexity, we use a different counting metric: the number of survivor sets that allow a quorum to be obtained. More carefully,

Definition 2. *Let $\langle P, \mathcal{S}_P \rangle$ be a system profile and $\mathcal{Q}$ be a coterie over P. The availability of $\mathcal{Q}$ is given by:* $\mathcal{A}(\mathcal{Q}) = |\{S : S \in \mathcal{S}_P \wedge S \notin \mathcal{T}(\mathcal{Q})\}|$

A coterie $\mathcal{Q}$ *covers* a survivor set S if there is a quorum $Q \in \mathcal{Q}$ such that $Q \subseteq S$. By the definition, $\mathcal{A}(\mathcal{Q})$ is hence the number of survivor sets that $\mathcal{Q}$ covers.

This is a good metric because in every step s of an execution E, there is at least one survivor set in $\mathcal{S}_P$ that does not intersect $F(E, s)$. If a coterie allows a quorum to be obtained for more survivor sets, then this coterie is available during more steps. As node vulnerability and edge vulnerability, $\mathcal{A}()$ is a deterministic metric and as such has a similar limitation with respect to probabilities. If we assign probabilities of failure to subsets of processes, then our metric may lead to wrong conclusions, as there might be higher available coteries that include discarded survivor sets. For the constructions and examples we discuss in this paper, however, using this metric gives us coteries with optimal availability.

[2] We use $x_1x_2 \ldots x_n$ as a short notation for the set $\{x_1, x_2, \ldots, x_n\}$.

If a coterie $\mathcal{Q}$ is dominated, then by definition there is some other coterie $\mathcal{Q}'$ that dominates $\mathcal{Q}$. Under reasonable assumptions, the availability of $\mathcal{Q}'$ is at least as high as the availability of $\mathcal{Q}$. Thus, we use domination to break ties between coteries that cover the same number of survivor sets. We say that $\mathcal{Q} \prec_a \mathcal{Q}'$ iff:

$$\bigvee \mathcal{A}(\mathcal{Q}') > \mathcal{A}(\mathcal{Q})$$
$$\bigvee (\mathcal{A}(\mathcal{Q}') = \mathcal{A}(\mathcal{Q})) \wedge \mathcal{Q}' \text{ dominates } \mathcal{Q}$$

In Section 5, we give quorum constructions that are optimal with respect to this metric without the tiebreaker rule. We do not discuss how to construct nondominated coteries from dominated ones; possible ways to do so are discussed in [5] and [8].

4 Failure models

In this section, we present two failure models that we use to derive quorum constructions. Both models are specific to multisite systems, and although they both model site failures, they model different system properties as we discuss in Section 6.

4.1 The multi-site hierarchical model

The first model, which we call the *multi-site hierarchical model*, decouples site failures from process failures. The failure model has two components: $\mathcal{F}_s$, which characterizes the failures of sites, and $\mathcal{F}_p$, which characterizes failures within a site. More specifically, $\mathcal{F}_s$ is a set of maximal subsets of sites that can fail simultaneously, $|\mathcal{F}_s| > 0$. $\mathcal{F}_p$ is an array with one entry for each site, where $\mathcal{F}_p[i]$ is the set of maximal subsets of processes that can be simultaneously faulty in site B_i when B_i is not faulty, $|\mathcal{F}_p[i]| > 0$, $i \in \{1, \ldots, |\mathcal{B}|\}$. Given an instance of this model, a set $S_i \subseteq P$ is in S_P if and only if:

$$\exists FS \in \mathcal{F}_s : \bigwedge \forall B_j \in \mathcal{B} \setminus FS : \exists FP \in \mathcal{F}_p[j] : B_j \cap S_i = B_j \setminus FP$$
$$\bigwedge \forall B_j \in FS : S_i \cap B_j = \emptyset$$

The multisite threshold model proposed in [16] is a threshold-based version of this model: f_s is the maximum number of sites that fail simultaneously, and $F_p[i]$ is the maximum number of processes that fail simultaneously in site B_i.

4.2 The bimodal model

The *bimodal model* is similar to the multisite hierarchical model: it also has two components $\mathcal{F}_s$ and $\mathcal{F}_p$. In general, this model represents settings in which there are multiple sites ($|\mathcal{B}| > 1$), all sites can fail but one, and if only one site is not faulty, then all processes in it are correct. Thus, each site is a survivor set. If multiple sites are nonfaulty, then the nonfaulty sites can have faulty processes. We describe these process failures with $\mathcal{F}_s$ and $\mathcal{F}_p$. Finally, we assume that there exists at least one site B_i such that $B_i \notin FS$ for every $FS \in \mathcal{F}_s$. Although B_i is not in any element of $\mathcal{F}_s$, it can still fail in the case that there is one nonfaulty site B_j with no faulty processes, and $j \neq i$. This assumption is necessary to derive an optimal construction, as we explain in Section 5.2.

The bimodal model contains the same failure patterns as the multisite hierarchical model for the same components $\mathcal{F}_s$ and $\mathcal{F}_p$, but it contains $|\mathcal{B}|$ additional failure

patterns, one for each site B_i. More specifically, a set $S_i \subseteq P$ is a survivor set for an instance of this model if and only if:

$$\begin{array}{l} \bigvee \exists FS \in \mathcal{F}_s : \bigwedge \forall B_j \in \mathcal{B} \setminus FS : \exists FP \in \mathcal{F}_p[j] : B_j \cap S_i = B_j \setminus FP \\ \qquad\qquad\qquad\qquad \bigwedge \forall B_j \in FS : S_i \cap B_j = \emptyset \\ \bigvee \exists B_j \in \mathcal{B} : S_i = B_j \end{array}$$

To construct a proper set of survivor sets, we need to impose the following constraint: $\forall FS \in \mathcal{F}_s : (|\mathcal{B} \setminus FS| > 1) \wedge (\forall B_i \in FS : \mathcal{F}_p[i] \neq \{\emptyset\})$. Without this constraint, the set of survivor sets might not be minimal, violating minimality.

The bimodal model does not have the intuitive appeal of the multisite hierarchical model. Nonetheless, we argue in Section 6 that for at least twosite systems, it is practical. In addition, it has theoretical interest, which we describe in Section 5.

5 Quorum constructions

In this section, we use the failure models described to derive quorum constructions that are optimal with respect to the metric $\mathcal{A}()$. The first construction covers all survivor sets in $\mathcal{S}_P$ by using $\mathcal{S}_P$ itself. We provide a necessary and sufficient condition for this to hold. The other construction is for systems in which it is not possible to cover all survivor sets. This is important when survivor sets do not pairwise intersect. This construction is also optimal with respect to the metric $\mathcal{A}()$ except that the resulting coterie may be dominated.

5.1 Achieving optimal availability

Let $\langle P, \mathcal{S}_P \rangle$ be a system profile, and suppose that we use the multisite hierarchical model to determine $\mathcal{S}_P$. To cover all survivor sets in $\mathcal{S}_P$, it is necessary and sufficient that $\mathcal{F}_s$ and $\mathcal{F}_p$ satisfy the following property:

$$\begin{array}{l} \forall FS, FS' \in \mathcal{F}_s : \exists B_i \in \mathcal{B} : \bigwedge B_i \notin FS \\ \qquad\qquad\qquad\qquad\qquad\quad \bigwedge B_i \notin FS' \\ \qquad\qquad\qquad\qquad\qquad\quad \bigwedge \forall FP, FP' \in \mathcal{F}_p[i] : \exists p \in B_i : p \notin FP \wedge p \notin FP' \end{array}$$

In words, we require that there is at least one site shared between any two survivor sets, and within that site there is at least one process that is shared between the two survivor sets. To show that this property is necessary, suppose that this property is violated. That is, there are FS, FS' in $\mathcal{F}_s$ such that, for every $B_i \in \mathcal{B}$, at least one of the following holds: 1) $B_i \in FS$; 2) $B_i \in FS'$; 3) there are $FP, FP' \in \mathcal{F}_p[i]$ such that for every $p \in B_i$, either $p \in FP$ or $p \in FP'$. This implies that there are at least two disjoint survivor sets S and S' in $\mathcal{S}_P$. Now suppose by way of contradiction that there is a coterie $\mathcal{Q}$ that covers all survivor sets in $\mathcal{S}_P$, i.e., $\mathcal{A}(Q) = |\mathcal{S}_P|$. We then have that there is a quorum $Q \in \mathcal{Q}$ such that $Q \subseteq S$. Similarly, there is a quorum $Q' \in \mathcal{Q}$ such that $Q' \subseteq S'$. Thus, if $S \cap S' = \emptyset$, then $Q \cap Q' = \emptyset$. We conclude that $\mathcal{Q}$ cannot be a coterie because it violates the 2–Intersection property.

To see that the property is sufficient is straightforward: by the definition of survivor sets, no survivor set is strictly contained in another, and the intersection property is guaranteed by assumption.

If we use $\mathcal{S}_P$ as a coterie, then we have achieved the best possible value for our availability metric because it covers all the survivor sets (i.e., $\mathcal{A}(\mathcal{S}_P)$ has the maximum value of $|\mathcal{S}_P|$). Using all the sites in the system, however, may be unnecessary. For example, if the system satisfies k–Intersection for some $k > 2$, then we may be able to construct a coterie over fewer sites. [3] We illustrate this point with a threshold version of the multisite hierarchical model. Suppose that every set $FS \in \mathcal{F}_s$ has the same size $f_s \geq 0$, and that for every $B_i \in \mathcal{B}$ and every $FP \in \mathcal{F}_p[i]$, we have that $|FP| = t$ for some nonnegative integer t. Then, if $|\mathcal{B}| \geq 2f_s + 1$, we only need to select a subset $\mathcal{B}' \subseteq \mathcal{B}$ of $2f_s + 1$ sites. For each site $B_i \in \mathcal{B}'$, we select $2t + 1$ processes from B_i. A quorum is obtained by selecting a majority of processes from a majority of sites in $\mathcal{B}'$.

We call this construction *Qsite*. As an example, suppose that $|\mathcal{B}| = 4$, $f_s = 1$, and for each site B_i, we have that $|B_i| = 4$ and $t = 1$. We then use 3 sites, as $2f_s + 1 = 3$, and 3 processes from each site, as $2t + 1 = 3$. From the construction, a quorum in $\mathcal{Q}$ is hence composed of four processes, two from a site B_i and two from a site B_j, $i \neq j$. This system has nine processes, and so a majority would consist of five processes. For both majority and Qsite, the coterie is available as long as there are $f_s + 1 = 2$ nonfaulty sites. Majority, however, not only requires that two sites are nonfaulty, but also that at least one of the sites contains no faulty processes. A coterie generated by Qsite does not have this same constraint, and it is available as long as there are two nonfaulty sites, each nonfaulty site containing two nonfaulty processes. This happens because majority uses larger quorums, and it tolerates fewer process failures.

It is not hard to see that Qsite requires fewer processes compared to majority coteries, and that the difference increases with the value of f_s (see [16] for details). Using fewer processes in each quorum reduces the load handled by any particular process, if quorums are uniformly selected, and increases the total capacity of the system [21].

5.2 The bimodal construction

It may be the case that the set of survivor sets do not satisfy 2–Intersection, and so can not be used as a coterie. For example, in the bimodal model, for each site B_i, B_i is a survivor set, and since sites are disjoint, $\mathcal{S}_P$ is not a coterie.

One can construct a coterie from any $\mathcal{S}_P$, though, by simply discarding survivor sets until remaining sets satisfy 2–Intersection. This procedure clearly will terminate with a coterie since a single set is a coterie of one quorum. To obtain a coterie that is optimal with respect to $\mathcal{A}()$, we need to determine the minimal set $\mathcal{S} \subset \mathcal{S}_P$ such that $\mathcal{S}_P \setminus \mathcal{S}$ is a coterie. The problem of computing the minimum number of survivor sets that have to be removed from $\mathcal{S}_P$ to obtain a coterie, however, is in general NP–Complete. We present the proof in Appendix A.

[3] k–Intersection generalizes 2–Intersection, and states that all subsets of k quorums intersect.

Under the bimodal model, it is simple to determine which survivor sets to discard. Consider the following intersection property that we call k-bimodal Intersection, $k > 1$:

$$\forall \text{ distinct } S_1, S_2, \ldots, S_{k+2} \in \mathcal{S}_P : \bigvee \exists i, j \in [1,k] : S_i \cap S_j \neq \emptyset \\ \bigvee S_{k+1} \cap S_{k+2} \neq \emptyset$$

Assume $\langle P, \mathcal{S}_P \rangle$ follows the bimodal failure model. According to the model, it contains $|\mathcal{B}|$ survivor sets that are disjoint, one for each site $B_i \in \mathcal{B}$. Also by the failure model, there is a site B_i such that $B_i \notin FS$, for every $FS \in \mathcal{F}_s$. Let S_i be the survivor set consisting of the processes of B_i. If $\langle P, \mathcal{S}_P \rangle$ also satisfies k-bimodal Intersection, $k = |\mathcal{B}|$, then we know that any two survivor sets S_a, S_b in $\mathcal{S}_P \setminus \{S_1, S_2, \ldots S_k\}$ intersect, and that $S_i \cap S_a \neq \emptyset$ and $S_i \cap S_b \neq \emptyset$. Since this is true for any S_a and S_b, the set $\mathcal{Q}_\ell = \{S_i\} \cup (\mathcal{S}_P \setminus \{S_1, S_2, \ldots S_k\})$ is a coterie, and $\mathcal{A}(\mathcal{Q}_\ell) = |\mathcal{S}_P| - (k-1)$. This is clearly optimal, since all of the remaining $k-1$ survivor sets do not intersect S_i. Also, if $\langle P, \mathcal{S}_P \rangle$ does not satisfy k-bimodal Intersection, then there is no coterie that covers $|\mathcal{S}_P| - (k-1)$ survivor sets, as there is no subset of $\mathcal{S}_P$ of size $|\mathcal{S}_P| - (k-1)$ that pairwise intersect. We call this construction *Bsite*.

6 Failure models in practice

The failure models presented in Section 4 are abstract views of failures in a multisite system. In this section, we present probabilistic models that we use to extract the parameters of our failure models. First, we use data from a real system to argue why we believe site failures are common in multisite systems. In the remainder of the section, we discuss process failures for the two models we propose in this paper. For each model, we discuss a framework based on a Markov chain and illustrate with an example.

6.1 Site failures

To understand how sites fail in a multisite system, we studied the failure data of a particular system, the BIRN Grid [6,16]. We obtained monthly availability data for 15 BIRN sites from January 2004 through August 2004.[4] According to this data, a site becoming unavailable is surprisingly common. On average, each site did not have 100% availability during five of the eight months, and in any given month several sites had unplanned outages and became unavailable.

Figure 1 summarizes the availability of sites. For each month, we count the number of sites that had availability below some value α, for different values of α. We then compute the average across the eight months for each value. This average is what we plot in Figure 1. From the figure, on average over ten sites do not have 100% availability in a month.

In trying to determine what causes low monthly site availability, we identified a few reasons for a site to be unavailable, observed in BIRN sites, in TeraGrid sites [25],

[4] This data is consistently collected by the BIRN staff, and made available through their web page. Availability figures are based on active probing (via ping) and on notifications generated by the Storage Resource Broker (SRB) service.

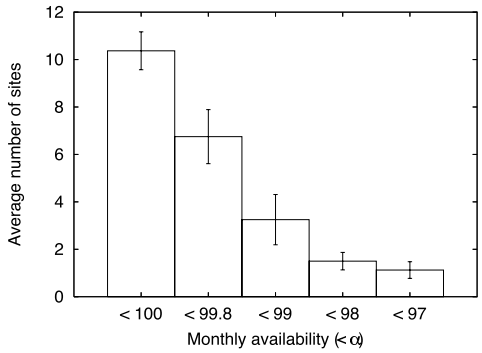

Fig. 1: Number of sites with availability below α. The error bars correspond to the standard error.

and in a local computer cluster. They are (in no particular order): Software problems; Power outages; Failure of shared resources (e.g. storage); Flooding resulting from broken pipes; Local campus network problems; Loss of aireonditioning. We are currently attempting to further quantify these failures.

6.2 Obtaining the multi-site hierarchical model

The multisite hierarchical model has two components: $\mathcal{F}_s$ that describes sites failures, and $\mathcal{F}_p$ that describes the failures of processes within a site. We can determine $\mathcal{F}_s$ using, for example, data such as described in Section 6.1. To determine $\mathcal{F}_p$, we need a model of failures within a site. Even when sites are not faulty, individual processes can fail due to, for example, hardware faults. In many multisite systems, hardware and software platforms are the same across the computing nodes (where processes run) of a site because of the difficulty in managing a heterogeneous environment. We hence assume that the reliability of processes within a site is uniform and independent. Of course, this assumption may be violated by viruses and worms [13], but their effects are outside the scope of this work.

We can model failures in sites using a *Markov chain* [24]. Instead of modeling the whole system, we have chosen to model sites individually. We assume that sites operate independently, and that outside of expected message communication the operation of a process at a site has little or no influence on the operation of a process at another site. As a consequence, sites change their failure states concurrently.

As process failures are independent, states of the model correspond to the number of faulty processes in a site, and the probability of undergoing a transition from a state with f faulty processes to a state with $f+1$ process is p. Repair transitions (from $f+1$ to f), however, may have probabilities that change with the value of f. For example, resources to repair processes can be progressively allocated as more processes fail. As a result, the repair probability remains constant or even increases with the value of f.

Figure 2 depicts the chain we just described. Assuming that no transition probability is zero, we have that this chain is irreducible and ergodic. According to the model,

processes fail independently, but the probability of repair (undergoing a transition from state $f+1$ to f) may change with the value of f. In our model, we use r_f to denote the probability that the site undergoes a transition from state $f+1$ to state f.

Repairs in different sites happen independently, and hence the probability of a repair transition does not increase with failures in different sites. That is, if a process fails in site B_i and another in site B_j, $i \neq j$, they do not mutually affect their repair probabilities.

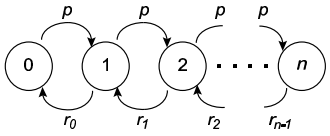

Fig. 2: Model for a single site with n process.

Using this model, we can easily compute a threshold on the number of failures for each site. First, we need to determine a target degree of reliability ρ, which is the probability that the number of simultaneous process failures in any site is higher than expected. Because our model is an irreducible ergodic Markov chain, we can compute the limiting probabilities of all states [24]. That is, the probability of being at a state j after a long time has elapsed, independent of the initial state i ($\pi_j = \lim_{n\to\infty} P_{ij}^n$). Using these limiting probabilities, we can determine a threshold for each site: the threshold for a site S_i is the number of failures associated to the first state that has a limiting probability smaller than ρ. This is implies that any state with failures above the threshold has probability lower than ρ.

To illustrate the process of obtaining a threshold for a site, we give an example. Let $\mathcal{B}$ be a collection of sites such that each site has three processes. Suppose that the probabilities of failure and repair are the same across all the sites. These probabilities are as follows: $p = 0.01$, $r_0 = 0.3$, $r_1 = 0.4$, and $r_2 = 0.5$. Computing the limiting probabilities, we have the following: $\pi_0 = 0.96695, \pi_1 = 0.03223, \pi_2 = 0.00080, \pi_3 = 0.00002$. If ρ is 0.001, for instance, we have that the threshold is one for every site, and $\mathcal{F}_p$ is as follows: $\mathcal{F}_p[i] = \{a_i : a_i \in B_i\}, \forall B_i \in \mathcal{B}$.

Note that the reverse order is also possible: choose a value for t and compute the corresponding probability of violating this threshold. Using one method or the other depends on design constraints.

6.3 Obtaining the bimodal model

From the description of the bimodal model in Section 4, when $k-1$ sites fail, the remaining site has no faulty processes. This means that the processes of each site comprise a survivor set. At the same time, it is possible that all available sites have faulty processes. We model this with a framework based on a Markov chain. Due to the complexity of this model, our framework is only meant to give a more practical view rather than serve as a general framework.

As in the previous section, the basic idea consists in determining probabilities for the possible states of the system, and to use a degree of reliability (a value $\rho \in [0, 1]$) to determine the states that we consider as normal states. Compared to the chain from the previous section, a state corresponds to failures across all the sites. We then label states with counters, one for each site. That is, we have one state for each possible value of the string $f_1 \cdot f_2 \cdot \ldots \cdot f_k$, where $0 \leq f_i \leq |B_i|$ and $B_i \in \mathcal{B}$. Using a directed graph as a way of visualizing the model, we have that the states are represented by nodes, and the

transitions by edges, where each edge has a weight that is the transition probability. In this model, we have three types of edges: sitefailure edges, processfailure edges, and repair edges. A sitefailure edge corresponds to the transition from a state in which a given site has one or more available processes to one in which all processes in this site are faulty. Using probability notation, let X_s be the random variable representing the state at step s. We then have that:

$$Pr\{X_{s+1} = f_1 \cdots |B_i| \cdot f_{i+1} \cdots f_k | X_s = f_1 \cdots f_i \cdot f_{i+1} \cdots f_k\} = p_s, f_i < |B_i| - 1$$
$$Pr\{X_{s+1} = f_1 \cdots |B_i| \cdot f_{i+1} \cdots f_k | X_s = f_1 \cdots |B_i| - 1 \cdot f_{i+1} \cdots f_k\} = p_f + p_s, f_i = |B_i| - 1$$

where p_f is the probability of a process failure, and p_s is the probability of a site failure. As a simplifying assumption, we have that p_s and p_f are constant across the sites. A processfailure edge is a transition from a state in which some site B_i has f_i faulty processes to a state in which B_i has $f_i + 1$ faulty processes, $f_i < |B_i|$. Using probability notation, we have:

$$Pr\{X_{s+1} = f_1 \cdots f_i + 1 \cdots f_k | X_s = f_1 \cdots f_i \cdots f_k\} = p_f, f_i < |B_i|$$

Finally, we call a repair edge a transition from a state in which $f_i + 1$ processes of some site B_i are faulty to a state in which f_i processes of B_i are faulty. That is:

$$Pr\{X_{s+1} = f_1 \cdots f_i \cdots f_k | X_s = f_1 \cdots f_i + 1 \cdots f_k\} = p_r(f_1 \cdots f_i + 1 \cdots f_k), f_i \geq 0$$

where $p_r()$ is a repair probability mapping. Different from p_s and p_f, we assume that the repair probability may differ for different states. In fact, this control over repair probabilities is what we use to guarantee that the properties of the bimodal model hold. An additional assumption that completes the model is that all other possible transitions have zero probability.

Figure 3 illustrates a model for two identical sites B_1 and B_2 of n processes each. In the figure, we mark the undesirable states by including them in a gray region. These states are the ones that violate the Bsite construction, and therefore must have low probability. To determine the probability of a state, we use also limiting probabilities.

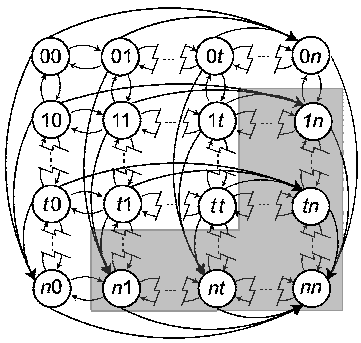

Fig. 3: Model for two sites.

In this model, we assume that probabilities of failure are constant, and they cannot be changed as the system changes states. We assume, however, that we are able to have different repair probabilities for different states. As a physical explanation, repair probabilities change as the effort spent to repair the system changes. Thus, we can increase the repair probability for an undesirable state, thereby decreasing the probability of being in this state. In practice, this means that the amount of physical resources used to repair processes must increase with the number of failures

in the system. It is then necessary to be able to detect failures. A failure detector for this application, however, can be unreliable as the only sideeffect is to have more resources used to repair processes unnecessarily. Having an unreliable failure detector implies that the repair probabilities have to take into account false positives. We therefore assume that it is possible to bound and estimate the frequency with which the failure detector makes mistakes.

As an example, suppose that $n = 3$, $p_s = 0.004$, $p_f = 0.001$, and $p_r(f_1 \cdot f_2) = 0.1$, if $f_1 \cdot f_4$ is outside the gray region, and $p_r(f_1 \cdot f_2) = 0.4$, if $f_1 \cdot f_2$ is inside the gray region. We have chosen these values using the following guidelines. First, as we observed in Section 6.1, site failures are common. We then assume that the probability of a site failure is higher than of a process failure, although we kept them in the same order of magnitude. Second, we assume that repair probabilities are much higher than the failure probabilities.

One still needs to choose a value for t, as repair probabilities depend upon this value. For such a small system, this choice is constrained to be either $t = 0$ or $t = 1$. If t is greater than 1, then the 2bimodal intersection property cannot be satisfied, and we are not able to construct a coterie using the technique proposed in Section 5. We therefore assume that $t = 1$, and we have the following limiting probabilities:

$$M = \begin{bmatrix} 0.7815 & 0.0391 & 0.0332 & 0.0344 \\ 0.0391 & 0.0020 & 0.0004 & 0.0004 \\ 0.0332 & 0.0004 & 0.0003 & 0.0004 \\ 0.0344 & 0.0004 & 0.0004 & 0.0004 \end{bmatrix}$$

where $M[f_1 + 1, f_2 + 1]$ is the limiting probability of state $f_1 \cdot f_2$. More specifically, we have that $\pi_{f_1 \cdot f_2} = M[f_1 + 1, f_2 + 1]$.

Suppose now that a_1, a_2, a_3 are the processes on one site, b_1, b_2, b_3 are the processes on the other site, and $4 \times 10^{-4} < \rho < 2 \times 10^{-3}$. We have that: 1) $\mathcal{F}_s = \{\emptyset\}$; 2) $\mathcal{F}_p[1] = \{a_i : a_i \in B_1\}$; 3) $\mathcal{F}_p[2] = \{b_i : b_i \in B_2\}$. The set of survivor sets is as follows:

$$\mathcal{S}_P = \{a_1a_2a_3, b_1b_2b_3\} \cup \{a_ia_jb_lb_m : i, j, l, m \in \{1, 2, 3\} \wedge i \neq j \wedge l \neq m\}$$

and from the Bsite construction, we have, for example, the following coterie:

$$\mathcal{Q} = \{a_1a_2a_3\} \cup \{a_ia_jb_lb_m : i, j, l, m \in \{1, 2, 3\} \wedge i \neq j \wedge l \neq m\}$$

which is dominated by:

$$\mathcal{Q}' = \{a_1a_2a_3\} \cup \{a_ib_jb_l : i, j, l \in \{1, 2, 3\} \wedge j \neq l\}$$

and we have by definition that $\mathcal{Q} \prec_a \mathcal{Q}'$. From the matrix M, observe that $\mathcal{Q}'$ is unavailable only in state 30, considering only allowed states (states that have probability greater than the degree of reliability). This is optimal as there is no coterie that is available for both states 30 and 03.

Although the system of the example is a simple one, it illustrates well that the bimodal model is implementable. We believe that the results can be generalized for two-site systems with more processes, but it is an open question whether there is a practical implementation for systems with more than two sites.

7 Evaluating coteries on PlanetLab

To evaluate the different choices for quorums in a multisite system, we conducted an experiment on PlanetLab using an implementation of the Paxos algorithm [18]. In brief, Paxos assumes that processes have one or more of the three following roles: Proposer, Acceptor, and Learner. Proposers propose ballots that are accepted by Acceptors. To propose, a Proposer has to read from and write to a quorum of Acceptors. Once an Acceptor accepts a ballot, it notifies the set of Learners. A Learner decides upon a value once it receives notifications from a quorum of Acceptors. More details in Appendix B.

In our experiment, we have three settings. In all settings, one single host (a UCSD host) has the roles of both a Proposer and a Learner, whereas the Acceptors are PlanetLab hosts spread across three sites (UC Davis, UT Austin, Duke). The settings are:

3Sites: One host from each site. A quorum consists of any set of two hosts. This is the Qsite construction, for $f_s = 1$, and $t = 0$;

3SitesMaj: Three hosts from each site. A quorum consists of majorities of hosts from two sites, and it has size four. This is the Qsite construction for $f_s = 1$, and $t = 1$;

SimpleMaj: Three hosts from each site. A quorum consists of any simple majority of sites. That is, any subset of five Acceptors.

For each setting, we have the Proposer issuing a new ballot every 15 minutes, and we log the time it takes to decide upon a value on this ballot. To implement a reliable channel, we create a new thread for every message sent, and this thread tries to send the message through a TCP connection until it succeeds. As a consequence, we have that every message sent by one process to another is eventually received, as long as the receiving process eventually recovers if it fails.

To register failures, every time establishing a TCP connection to another host times out, we log it to a file. A failure in this case is the inability to reach the Acceptor, not necessarily implying that the host has crashed. That is, the unavailability of a host may be caused by a network partition.

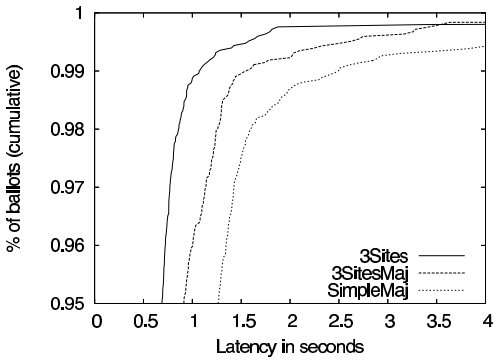

Fig. 4: Cumulative latency distribution.

We had these three settings running in parallel for 27 days in April 2005. Figure 4 shows part of the cumulative distribution function for the latency of reaching agreement on each ballot. We also show in Table 1 the percentage of samples with value greater than $4s$.

It is not surprising that 3Sites has best response time for the average case, followed by 3SitesMaj and SimpleMaj, since quorums have fewer Acceptors in this exact order. However, the graph shows that there is a point (around $3.5s$) in which the curve for 3SitesMaj crosses 3Sites. This implies that there are fewer samples for 3SitesMaj with latency greater than $3.5s$ than for 3Sites. As the tail of the distribution for 3SitesMaj contains fewer samples, it has best availability among the three in this experiment.

To understand why this is the case, we need to understand what components are involved in deciding upon a ballot. The latency of a ballot has two main components: message latency and process failures. From the graph, the message latency component dominates until $3.5s$. After $3.5s$, the delay is mostly caused by the inability to reach enough Acceptors. Having more processes increases the latency for 3SitesMaj and SimpleMaj compared to 3Sites for values under $3.5s$, where the message latency component has more weight. On the other hand, 3SitesMaj presents better response time for values greater than $3.5s$, when there are process failures. Thus, there is a tension between obtaining good response time on average and having a larger percentage of the samples within a bounded response time. This information is important, for example, when determining the timeout for a quorum-based service. Considering our three settings, if a timeout value greater than $3.5s$ is chosen, then 3SitesMaj is likely to time out less often than 3Sites and SimpleMaj. Finally, an interesting observation is that SimpleMaj not only had the worst average response time, but also had the largest percentage of samples with response time greater than $4s$. This indicates that using majority quorums is a poor choice for multisite systems.

We also counted the number of ballots for which the Proposer could not initially contact enough Acceptors to obtain a quorum, and the decision on the ballot was therefore delayed until enough Acceptors were available. When this happens to a ballot, we say that this ballot is *postponed*. For each setting, we have the following:

	Latency $> 4s$ **(%)**
3Sites	0.0020
3SitesMaj	0.0016
SimpleMaj	0.0057

Table 1: Samples with value greater than 4 seconds.

3Sites: There were 3 postponed ballots;

3SitesMaj: There were 2 postponed ballots. Only for one of these ballots, there would be one quorum available in the simple majority scheme;

SimpleMaj: There were 4 postponed ballots. For all these ballots, using the majorities of two sites would give us an available quorum.

The data presented in this section is perhaps not conclusive because the number of failures observed was too small to be statistically valid. Moreover, PlanetLab is not a production system in the sense that sites are not designed to be highly available, and node repair is often leisurely. On the other hand, the results presented do not contradict any of our assumptions, thus indicating that our models may be suitable even for multisystems such as PlanetLab.

8 Conclusions

This paper is a first step into the practical construction of coteries for multisite systems. We base one coterie construction on a failure model that we motivate from failure measurements from a deployed multisite system and from a Markov model. We also consider a weaker failure model that has some theoretical and practical interest. We define optimality by introducing a metric that is suitable to dependent failures, and we show that our quorum constructions are optimal with respect to this metric.

Being a first step, this paper leaves some questions unanswered. First, our multisite hierarchical model is intuitive and is based on some failure data from a real system. How typical is this system? Is the model broadly applicable? Second, our bimodal model is based on the idea of having different repair probabilities for different states. This technique, which essentially integrates operating procedures with the failure model, appears to be a potentially powerful new direction for the design of novel and efficient protocols. Finally, we describe a method of building a coterie from survivor sets that do not satisfy 2-Intersection. The survivor sets are defined by some target availability, and the availability of the quorum system is reduced by discarding survivor sets. How does this strategy compare with one in which the initial target availability is increased until the survivor sets satisfy 2-Intersection?

References

1. Y. Amir and A. Wool. Evaluating quorum systems over the Internet. In *Proceedings of the 26th IEEE FTCS*, pages 26–37, Sendai, Japan, June 1996.
2. Y. Amir and A. Wool. Optimal availability quorum systems: Theory and practice. *Information Processing Letters*, 65(5):223–228, Mar. 1998.
3. D. Barbara and H. Garcia-Molina. The vulnerability of vote assignments. *ACM Transactions on Computer Systems*, 4(3):187–213, Aug. 1986.
4. M. Bellare and S. Goldwasser. The complexity of decision versus search. *SIAM Journal on Computing*, 23(1):97–119, Feb. 1994.
5. J. Bioch and T. Ibaraki. Generating and approximating nondominated coteries. *IEEE Transactions on Parallel and Distributed Systems*, 6(9):905–914, Sept. 1995.
6. The Biomedical Informatics Research Network (BIRN). http://www.nbirn.net.
7. J.-M. Busca, M. Bertier, F. Belkouch, P. Sens, and L. Arantes. A performance evaluation of a quorum-based state-machine replication algorithm for computing grids. In *Proceedings of the 16th IEEE SBAC-PAD'04*, Foz do Iguaçú, PR, Brazil, Oct. 2004.
8. H. Garcia-Molina and D. Barbara. How to assign votes in a distributed system. *Journal of the ACM*, 32(4):841–860, Oct. 1985.
9. M. R. Garey and D. S. Johnson. *Computers and Intractability: A guide to the theory of NP-Completeness*. W. H. Freeman and Company, 1979.
10. D. Gifford. Weighted voting for replicated data. In *Proceedings of ACM SOSP*, pages 150–162, Pacific Grove, CA, USA, Dec. 1979.
11. S. Gilbert and G. Malewicz. The Quorum Deployment Problem. In *Proceedings of OPODIS*, pages 218–228, Grenoble, France, Apr. 2004.
12. E. Halperin. Improved approximation algorithms for the vertex cover problem in graphs and hypergraphs. *SIAM Journal on Computing*, 31(5):1608–1623, Feb. 2002.
13. F. Junqueira, R. Bhagwan, A. Hevia, K. Marzullo, and G. M. Voelker. Surviving Internet catastrophes. In *Proceedings of USENIX Tech. Conference, General Track*, pages 45–60, Anaheim, CA, USA, Apr. 2005.
14. F. Junqueira and K. Marzullo. Synchronous consensus for dependent process failures. In *Proceedings of the 23rd IEEE ICDCS*, pages 274–283, Providence, RI, USA, May 2003.
15. F. Junqueira and K. Marzullo. Replication predicates for dependent-failure algorithms. In *Proceedings of the 11th Euro-Par Conference*, LNCS 3648, pages 617–632, Lisbon, Portugal, Aug. 2005.
16. F. Junqueira and K. Marzullo. The virtue of dependent failures in multisite systems. In *Proceedings of the IEEE Workshop on Hot Topics in System Dependability*, Supplemental volume of DSN'05, pages 242–247, Yokohama, Japan, June 2005.

17. A. Kumar. Hierarchical Quorum Consensus: A new algorithm for managing replicated data. *IEEE Transactions on Computers*, 40(9):996–1004, Sept. 1991.
18. L. Lamport. The part-time parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, May 1998.
19. L. Lamport. *Specifying systems: The TLA+ language and tools for hardware and software engineers*. Addison-Wesley, 2002.
20. M. Maekawa. A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. *ACM Transactions on Computer Systems*, 3(2):145–159, May 1985.
21. M. Naor and A. Wool. The load, capacity, and availability of quorum systems. *SIAM Journal on Computing*, 27(2):423–447, Apr. 1998.
22. D. Peleg and A. Wool. Crumbling Walls: A class of practical and efficient quorum systems. In *Proceedings of ACM PODC*, pages 120–129, Ottawa, Ontario, Canada, Apr. 1995.
23. The Planetlab testbed. http://www.planet-lab.org/.
24. S. Ross. *Introduction to probability models*. Harcourt Academic Press, 2000.
25. The TeraGrid project. http://www.teragrid.org/.
26. H. Yu. Signed Quorum Systems. In *Proceedings of the 23rd ACM PODC*, pages 246–255, St. John's, Newfoundland, Canada, July 2004.

A NP-Completeness of the DSS problem

In this section, we show that the problem of determining the minimum number of survivor sets that has to be removed from a set S_P such that the remaining survivor sets form a coterie is NP-Complete. Our strategy consists in defining a decision problem, and showing that this problem is NP-Hard with a reduction from the Vertex-Cover problem. We then argue that an algorithm for the decision problem can be used to solve the search problem, which consists in determining the actual set of survivor sets to remove. Finally, we argue that we can use a search solution to solve the optimization problem. The optimization problem consists in determining a minimal set of survivor sets to be removed.

We now define the DSS (Disjoint Survivor Sets) decision problem. Instead of assuming only a set of survivor sets S_P as the input of the problem, we also assume pairs of survivor sets in S_P that are disjoint. To compute such pairs from the set of survivor sets S_P in a system profile $\langle P, \mathcal{S}_P \rangle$ can be done in polynomial time by simply checking the intersection of every pair of survivor sets. A simple solution for this algorithm has time complexity $O(|P| \cdot |S_P|)$. This simple algorithm consists in having a matrix M with $|P|$ rows and $|S_P|$ columns. For each process p, we set $M_{p,s}$ if the survivor set associated with column s contains process p. Another pass on the matrix provides the information we are looking for. This algorithm is perhaps not the best one to use. A further discussion of algorithms to optimally find such disjoint sets is out of the scope of this paper.

We then state the DSS decision problem as follows:

Instance : A set S_P of survivor sets, a set $\mathcal{P}$ of pairs of survivor sets in S_P, and a positive integer $r \leq |S_P|$.
Question : Is there a subset $\mathcal{S} \subseteq S_P$ such that:
1. For every $(S_1, S_2) \in \mathcal{P}$: $S_1 \in \mathcal{S} \vee S_2 \in \mathcal{S}$;
2. $|\mathcal{S}| \leq r$.

Consider now the definition of the VC (Vertex Cover) problem, extracted from [9]:

Instance : A graph $G = (V, E)$ and a positive integer $k \leq |V|$.
Question : Is there a vertex cover of size k or less for G, that is, a subset $V' \subseteq V$ such that $|V'| \leq k$, and for each edge $\{u, v\} \in E$, at least one of u and v belongs to V'?

The following two claims show that DSS is NP-complete.

Claim. VC $\leq_m$ DSS

Proof. To show this claim, we have to provide a polynomial-time algorithm that outputs an instance $\langle S_P, \mathcal{P}, r\rangle$ of the DSS problem given an instance $\langle G, k\rangle$ of the VC problem such that:

$$\langle G, k\rangle \in \text{VC} \rightarrow \langle S_P, \mathcal{P}, r\rangle \in \text{DSS} \quad (1)$$

$$\langle S_P, \mathcal{P}, r\rangle \in \text{DSS} \rightarrow \langle G, k\rangle \in \text{VC} \quad (2)$$

Consider the following algorithm:

Algorithm VCtoDSS on input $\langle G = (V, E), k\rangle$

$S_P \leftarrow \emptyset$
$\mathcal{P} \leftarrow \emptyset$
For every $v \in V$:
    $S_P \leftarrow S_P \cup \{S_v\}$
For every edge $\{u, v\} \in E$:
    $\mathcal{P} \leftarrow \mathcal{P} \cup \{(S_u, S_v)\}$
$r \leftarrow k$
output $\langle S_P, \mathcal{P}, r\rangle$

The algorithm clearly runs in polynomial time. We then need to show implications 1 and 2. First we show 1. Suppose that $\langle G, k\rangle \in$ VC. We then have that there is a vertex cover V' of size at most k for G. That is, there is a subset V' of V of size at most k such that for every edge $\{u, v\}$, either u or v is in V'. By the construction of the algorithm, if $\mathcal{S}$ is the set of all S_v such that $v \in V'$, then $\mathcal{S}$ must be such that for all $(S_v, S_u) \in \mathcal{P}$, either $S_v \in \mathcal{S}$ or $S_u \in \mathcal{S}$. We conclude that $\langle S_P, \mathcal{P}, r\rangle$ is in DSS.

It remains to show implication 2. Suppose that $\langle S_P, \mathcal{P}, r\rangle$ is in DSS. This means that there is a subset $\mathcal{S}$ of S_P such that $|\mathcal{S}| \leq r$ and for every $(S_v, S_u) \in \mathcal{P}$, either $S_v \in \mathcal{S}$ or $S_u \in \mathcal{S}$. We then have that $V' = \{v : S_v \in \mathcal{S}\}$ must be a vertex cover for G, and $|V'| \leq k$, by the construction of the algorithm. This concludes the proof of our claim.

With a simple modification of the algorithm presented in the previous proof, we can also show that DSS map-reduces to VC. This is important because there are efficient approximation algorithms for the vertex cover problem, such as the ones in [12], that can be used to compute a solution for DSS.

Claim. DSS is in NP

Proof. We need to provide a polynomial-time verifier that takes an instance $\langle S_P, \mathcal{P}, r\rangle$ of the DSS problem and a certificate C. The verifier then outputs whether $\langle S_P, \mathcal{P}, r\rangle \in$ DSS or not. The certificate consists of a subset of S_P. We have that the verifier works as follows:

Verifier on input $\langle S_P, \mathcal{P}, r\rangle, C$
  Parse C into set $\mathcal{S}$
  Verify if $|\mathcal{S}| \leq r$
  Verify if $\mathcal{S} \subseteq S_P$
  Verify if for every $(S_i, S_j) \in \mathcal{P}$,
    either $S_i \in \mathcal{S}$ or $S_j \in \mathcal{S}$
  If any verification fails return false, else return true

The verifier clearly runs in polynomial time on the size of the input.

A well-known result from complexity theory says that search reduces to decision for NP-complete problems [4]. For DSS the search problem consists of finding a subset $\mathcal{S}$ of S_P such that $|\mathcal{S}| \leq r$ and for every $(S_i, S_j) \in \mathcal{P}$, either $S_i \in \mathcal{S}$ or $S_j \in \mathcal{S}$. The algorithm returns $\perp$ if no such a set exists. Thus, we can only have a polynomial-time algorithm for the search problem if there is a polynomial-time algorithm for the decision problem. Such an algorithm only exists if P = NP . To find the smallest set $\mathcal{S}$ given a system profile $\langle P, \mathcal{S}_P\rangle$ we can run a search algorithm for smaller values of r, and return the set $\mathcal{S}$ returned by the smallest value of r for which the search algorithm does not return $\perp$. We then also have that there is a polynomial-time algorithm for the optimization problem if and only if P = NP .

B The Paxos algorithm

The classic Paxos algorithm requires two rounds of messages. First, a Proposer determines a ballot number b and sends a propose message to the Acceptors containing b. Once an Acceptor receives such a message, it sends a promise message back if it has not promised not to accept ballots with values $b' \leq b$. Also, a promise message contains the last value accepted by the Acceptor, if any. Once it receives promises from a quorum of Acceptors, the Proposer chooses a value v with the highest ballot number accepted by any Acceptor, or its own value if there is no such a v, and sends accept messages to Acceptors containing v and b. Upon reception of an accept message, an Acceptor accepts this value by updating the values for the last accepted ballot and the last accepted value. The Acceptor also sends learn messages to Learners, informing of its decision.