Evaluating Quorum Systems over the Internet

Yair Amir
Department of Computer Science
The Johns Hopkins University
Baltimore MD 21218
and the NASA Center of Excellence
in Space Data and Information Sciences
yairamir@cs.jhu.edu

Avishai Wool
Department of Applied Mathematics
and Computer Science
The Weizmann Institute of Science
Rehovot 76100, Israel
yash@wisdom.weizmann.ac.il

Abstract

Quorum systems serve as a basic tool providing a uniform and reliable way to achieve coordination in a distributed system. They are useful for distributed and replicated databases, name servers, mutual exclusion, and distributed access control and signatures.

Traditionally, two basic methods have been used to evaluate quorum systems: the analytical approach, and simulation. This paper proposes a third, empirical approach. We collected 6 months' worth of connectivity and operability data of a system consisting of 14 real computers using a wide area group communication protocol. The system spanned two geographic sites and three different Internet segments.

We developed a mechanism that merges the local views into a unified history of the events that took place, ordered according to an imaginary global clock. We then developed a tool called the Generic Quorum-system Evaluator (GQE), which evaluates the behavior of any given quorum system over the unified, real-life *history.*

We compared fourteen dynamic and static quorum systems. We discovered that as predicted, dynamic quorum systems behave better than static systems. However we found that many assumptions taken by the traditional approaches are unjustified: crashes are strongly correlated, network partitions do occur, even within a single Internet segment, and we even detected a brief simultaneous crash of all the participating computers.

1. Introduction

1.1. Motivation

Quorum systems serve as a basic tool providing a uniform and reliable way to achieve coordination between processors in a distributed system. Quorum systems are defined as follows. A *set system* is a collection of sets $\mathcal{S} = \{S_1, \ldots, S_m\}$ over an underlying universe $U = \{u_1, \ldots, u_n\}$. A set system is said to satisfy the *intersection property*, if every two sets $S, R \in \mathcal{S}$ have a nonempty intersection. Set systems with the intersection property are known as *quorum systems*, and the sets in such a system are called quorums.

Quorum systems have been used in the study of distributed control and management problems such as *data replication protocols* (cf. [11, 17, 3, 23]), *name servers* (cf. [28]), *mutual exclusion* (cf. [36]), *selective dissemination of information* (cf. [39]), and distributed access control and signatures (cf. [29]).

A protocol template based on quorum systems works as follows. In order to perform some action (update the database, say), the user selects a quorum and *accesses all its elements*. The intersection property then guarantees that the user will have a consistent view of the current state of the system.

For example, if all the members of a certain quorum give the user permission to enter the critical section, then any other user trying to enter the critical section before the first user has exited (and released the permission-granting quorum from its lock) will be refused permission by at least one member of any quorum it chooses to access.

1.2. Evaluating Quorum Systems

Traditionally, two basic methods have been used to evaluate quorum systems:

- **Analysis:** In this approach the optimal quorum system is computed, using some stochastic model. Usually, assumptions such as independent failures and perfect communication are made to render the problem tractable. The results of this approach are rigorous and definitive. Analyses of the availability (probability that a live quorum exists in the system), assuming a complete and fault free network, can be found in [8, 34, 12, 37, 7]. Analyses of the expected size of a connected component containing a quorum, and the availability on ring networks are [32, 19]. An analysis of the load can be found in [30, 18]. A queuing system analysis is [26].

- **Simulation:** In this approach (cf. [21, 14, 24]) a simulation model is constructed, and a simulation is run. This approach allows more complex models, which can not be analyzed completely. Usually, much stronger assumptions are made, such as failure distribution, mean-time to repair, etc. These assumptions are embedded in the simulation model.

1.3. New Results

This paper proposes an empirical approach. We collected 6 months' worth of connectivity and operability data of a system consisting of 14 real computers using a wide area group communication protocol. The system spanned two geographic sites and three different Internet segments. Each computer recorded to a local log file every change in the membership, i.e., in the set of other machines it was currently connected to. Local recoveries and crashes were recorded to the local log as well. Each log record was timestamped with the local time.

We developed a mechanism that merges all the local files into a unified history of the events that took place, ordered according to an imaginary global clock. This non-trivial mechanism had to overcome inconsistent local views, unsynchronized clocks and operator errors.

We then developed a tool called the Generic Quorum-system Evaluator (GQE), which evaluates the behavior of any given quorum system over the unified history. The GQE lets us compare the performance of different quorum systems over the same *real-life* history of events. We compared fourteen dynamic and static quorum systems, some of which are predicted to behave optimally under different theoretical models, and other ad-hoc ones. These quorum systems were programmed according to the GQE template, which can be easily modified to accommodate new structures of quorum systems. The GQE, the quorum systems and the unified history file are all publicly available.

We discovered that as predicted, dynamic quorum systems behave somewhat better than static systems. This result was validated for several variants of both static and dynamic systems. Quorum systems that are predicted to have optimal availability assuming full connectivity were found to be inferior to systems that are confined to a single biconnected component.

Moreover, we found that many assumptions taken by the traditional approaches are unjustified: crashes are strongly correlated, and network partitions do occur, even within a single Internet segment. In one case, we even detected a simultaneous crash of all the participating computers (during a wide-area power failure).

As a side effect, we observed significant clock drifts between the different machines in our experiment, which we had to overcome in our history unification mechanism.

Of course, these results represent the events that occurred in our network. We hypothesize, though, that they are typical of current networks technology. We plan to test this hypothesis by launching a larger scale experiment, involving around ten sites, in the near future. The infrastructure we have already developed can be easily utilized in a more complex environment.

The remainder of the paper is organized as follows. Section 2 goes over some of the theoretical analysis models. Section 3 describes the data collecting experiment and the developed software packages: the wide-area group communication, the unifying program, and the Generic Quorum-system Evaluator. Section 4 describes the observed failure patterns, and Section 5 details quorum evaluation results. Section 6 describes the clock drift we observed. Section 7 concludes the paper.

2. Theory

2.1. Basic Definitions

A formal definition of a quorum system is the following.
Definition 2.1 *A* Set System $\mathcal{S} = \{S_1, \ldots, S_m\}$ *is a collection of subsets* $S_i \subseteq U$ *of a finite universe* U *representing the processors. A* Quorum System *is a set system* $\mathcal{S}$ *that has the* Intersection property: $S \cap R \neq \varnothing$ *for all* $S, R \in \mathcal{S}$. *The sets of the system are called* quorums.

Many quorum systems which are based on combinatorial constructions appear in the literature, such as [25, 13, 1, 10, 35]. However all the systems with optimal availability (under the fully connected model, see Section 2.3.1) turn out to be defined by voting.

Definition 2.2 *Let* v_i *be an integer vote assigned to processor* i, *and let* $V = \sum_i v_i$. *The* voting system *defined by the votes* v_i *is the collection of all the sets which have more than half the total vote, i.e., all* $S \subseteq U$ *s.t.* $\sum_{i \in S} v_i > V/2$.

2.2. Quality Measures

We are interested in measures that quantify the quality of service that a particular quorum system provides. A basic notion in these measures is that of a *live quorum*: a quorum is called alive when all its processors are alive. The measures we consider are:

UnAvail**:** This is the *un-availability* of the system [8], the probability of the event that no live quorum exists in the system. When such an event occurs the service is completely halted. The un-availability is widely accepted as the measure by which quorum systems are evaluated.

AvgUnAcc**:** Another appropriate measure [22] is the *un-accessibility* UA_i of a processor i, i.e., the probability that the network component in which i resides does not contain a live quorum, given that i is alive. The average un-accessibility is:

$$\mathrm{AvgUnAcc} = \sum_i UA_i \frac{\Pr(i \text{ alive})}{\sum_j \Pr(j \text{ alive})}.$$

WorstUnAcc**:** This is the worst un-accessibility incurred by any processor: $\mathrm{WorstUnAcc} = \max_i\{UA_i\}$. This represents the worst level of service experienced by any one processor.

2.3. A Taxonomy of Optimal Quorum Systems

The optimality of a quorum system depends on the failure model that is assumed. Several such models have been analyzed, and all share the following assumptions:

- Independence: Processors failures are independent, and so are network links failures.
- Fail-Stop: A failed processor stops to function rather than functions incorrectly.
- Transient: The failures are detectable and are repaired.

2.3.1 Fully Connected

The simplest model is one in which it is assumed that the communication network is fully connected, and the network links never fail. Therefore in such a model the network is never partitioned into disconnected components. Within this model there are two sub-models:

Uniform: The processors fail with the same, fixed probability p. If $p < 1/2$ then [8] shows that the the majority system (i.e., a voting system with $v_i = 1$ for all processors) has optimal availability. If $p > 1/2$ then [34, 12] show that a monarchy (primary site) system is optimal.

Non-Uniform: The processors have different failure probabilities p_i. Then [37, 7] show that the optimal availability quorum system is a voting system in which the votes depend on the failure probabilities p_i as follows. If $p_i \geq 1/2$ for all i then a monarchy with the least failprone processor acting as king is optimal. Otherwise, set $v_i = 0$ for every processor with $p_i \geq 1/2$, and for all other processors set

$$v_i = \log_2 \frac{1 - p_i}{p_i}. \qquad (1)$$

2.3.2 Partially Connected

In this model the communication network may have an arbitrary topology, and the network links may fail in addition to the processor failures. Therefore in such a model the network can partition into several disconnected components.

In this model [8] show that the optimal availability quorum system is always contained within a single biconnected component of the network, for any topology. For a ring topology [32, 19] give explicit optimal solutions.

2.3.3 Dynamic Quorums

All the above mentioned optimal quorum systems are static. A more general type of system [16] is a dynamic one in which the quorums are defined adaptively over time (e.g., the vote of each processor may change).

In [20] it is shown that dynamic voting has better availability than any static system, when the processors' time-to-failure and the time-to-repair have an exponential distribution (in addition to the independence assumptions) for fully connected networks.

3. Methods

3.1. The Network

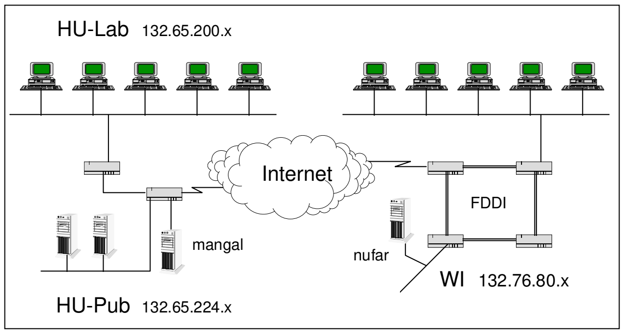

Figure 1. The physical network layout.

Our experimental system spans two geographical sites in Israel. The first site resides in the Institute of Computer Science of the Hebrew University of Jerusalem (`cs.huji.ac.il`), and the second in the Department of Applied Mathematics and Computer Science of the Weizmann Institute of Science (`wisdom.weizmann.ac.il`). The two sites are connected by a multi-hop connection over the Internet. The distance between the two sites is about fifty kilometers.

The network is composed of three logical Internet segments. Each segment represents a direct logical connection between any two machines on that segment. In particular, a broadcast message multicast on a segment may, in principle, be received by all its members. The logical layout is as follows.

HU-Lab Five machines in the Transis project lab: hazard (132.65.200.21),
havoc (132.65.200.22), hashem (132.65.200.23), harpo (132.65.200.24) and hal (132.65.200.10).

HU-Pub Three departmental servers used by students at the Institute of Computer Science: pita (132.65.224.9), bagel (132.65.224.10) and mangal (132.65.224.20).

WI Five general purpose machines used by students at Weizmann: marva (132.76.80.76), al (132.76.80.90), kalanit (132.76.80.81), nurit (132.76.80.56), hadar (132.76.80.17), and one departmental server: nufar (132.76.80.52).

In the past, each logical Internet segment was typically mapped to a single physical network such as Ethernet, Token Ring, or FDDI. Today, however, it is not rare to find logical segments which are composed of several physical segments connected by switching or bridging elements. Although the user may not be aware of this structure, it invalidates the basis for the common assumption that network partitions do not occur inside a single logical segment.

Indeed, as can be seen in Figure 1, of our three segments only the HU-Lab segment is actually a physical segment (Ethernet) as well. The HU-Pub segment is composed of two physical Ethernet segments connected by switching devices. The WI segment is even more complex, connecting two Ethernet segments over an FDDI backbone. Our records show many intra-segments partitions which occurred according to this physical structure.

3.2. The Protocol

As a wide area group communication mechanism, we used the Spread [2] reliable multicast tool. Spread is run as a daemon on each of the machines participating in the experiment, and is automatically activated when the machine boots. The Spread daemons are used to keep track of the membership lists of the dynamically changing connected network components. Spread complies with the *extended virtual synchrony* model [27], developed in the Totem [6] and Transis [5] projects. This model defines consistent semantics for services across all the daemons in a system that is prone to network partitions and processor crashes.

Each Spread daemon logs three kinds of records to a local file: "B" (boot), "A" (active), and "M" (membership change). Every record is time-stamped with the local machine clock.

A "B" record simply indicates the machine's boot. An "A" record indicates that the machine observed no change in the membership, and that it is still active. The protocol writes an "A" record to the end of the local file every T_A minutes (we used $T_A = 10$), which *is replaced* by the next "A" or "M" record. This allows us to detect real machine crashes: a crash appears as the last "A" record (written at most T_A minutes before the crash) followed by a "B" record when the machine recovers.

The "M" records indicate a change in the membership, as observed by the current Spread daemon. During every connectivity state of the network, Spread defines a *component leader* in every connected component of the network. The component leader is the machine with the smallest ID according to the order in Table 1. Therefore an "M" record contains the current machine's leader ID, the list of all the machines in this component, and the value of the leader's clock when it detected the change. This leader time serves as a membership change identifier that is *agreed* upon by all the members in the connected component.

The algorithm used by Spread to determine the membership is a version of the Transis membership algorithm [4], modified to cope with the wide-area structure of a network which connects several local area broadcast domains. Like Transis and Totem, Spread makes use of the non-reliable multicast service inside the local Internet segments, when it is available.

3.3. The Unification Mechanism

The Unification Mechanism produces a unified file representing the connectivity and operability events which occurred in the system. It starts with an empty unified history log file and combines the local log files one by one. In each round it produces a refined history log which incorporates the view of the current local log, merged into the previous unified file. After all the local log files are processed, the unified file represents all the logged crashes, recoveries, network partitions and merge events that occurred in the system during the experiment.

An important task of the unifying mechanism is to identify that "M" records from different machines represent the same membership change event. The basis by which it makes this decision is the fact that the record of every machine in the current component has the same membership identifier (i.e., the leader time-stamp). This identifier is determined by the group communication protocol according to the extended virtual synchrony model specification.

A crash is detected using the "A" and "B" record combination, as noted above. However the last "A" record may have been written up to 10 minutes before the crash. Therefore for computing the *length* of a crash the unifying mechanism tries first to rely on observations made by other machines who sensed the crash.

The global clock is an imaginary clock that runs at the rate of hazard's clock (the machine with ID 0). Local

Record type	Local time	Leader time	0	1	2	3	4	5	6	7	8	9	10	11	12	13
M	797996823	797996823	0	0	0	0	–	0	0	0	0	0	0	0	0	0
M	798024996	798024996	0	0	0	0	–	0	0	0	–	–	–	–	–	–
M	798026834	798026834	0	0	0	0	–	0	0	0	–	0	–	–	0	0
M	798026900	798026900	0	0	0	0	–	0	0	0	–	0	0	0	0	0

Figure 2. Hazard's local file.

Global time	Leader time	Leader	0	1	2	3	4	5	6	7	8	9	10	11	12	13
797996823	797996823	0	0	0	0	0	**	0	0	0	0	0	0	0	0	0
798024996	798024996	0	0	0	0	0	**	0	0	0	8	8	8	8	8	8
798025004	798025102	8	0	0	0	0	**	0	0	0	8	8	8	8	8	8
798026458	798026566	9	0	0	0	0	**	0	0	0	**	9	**	**	9	9
798026834	798026834	0	0	0	0	0	**	0	0	0	**	0	**	**	0	0
798026900	798026900	0	0	0	0	0	**	0	0	0	**	0	0	0	0	0

Figure 3. The unified history file.

records of events that take place in hazard's connected component are time-stamped with hazard's clock value, so this is the computed global time for them. Other events receive an estimated global time that is based on the local clock and the last known difference between the current component leader's clock and that of hazard.

Figure 2 shows an excerpt from the local file of hazard (ID 0), covering 30077 seconds (about 8:21 hours). Hazard was alive during this period hence all the records are of type "M". Dashes mark machines unreachable from hazard, either due to a machine crash or a network partition.

Figure 3 shows an excerpt from the unified history, covering the same sequence of events. Asterisks denote crashed machines. From the global point of view we see that hal (ID 4) is down throughout the period. We also see that the WI segment (IDs 8–13) becomes disconnected from the HU machines, then unknown to hazard, 3 machines at WI crash simultaneously, then the network merges back into a single component, and finally two WI machines recover.

3.4. The Generic Quorum-System Evaluator

The Generic Quorum-system Evaluator (GQE) is a program that analyses the behavior of any given quorum system over a unified history log file. The GQE defines an interface, composed of two routines:

`Init_quorum()`: This routine initializes any internal data structures.

`Check_quorum( membership, has_access )`: This routine gets a membership list and calculates whether a live quorum exists, and whether each machine's connected component contains a live quorum. For static quorum systems, the results depend only on the given membership list. For dynamic systems, they depend on the given membership list, and on the previous history (as determined by the internal data structures).

After the initialization, the GQE feeds the `Check_quorum` routine with the membership changes which occurred in the network according to the unified history log. The GQE keeps track of which processors have access to a live quorum and for how long. Based on this information, the GQE computes the quality measures we defined in Section 2.2, for the given quorum system.

We have programmed packages conforming to this interface specification for each of the 14 quorum systems we evaluated. Each package contains less than 100 lines of code, indicating the simplicity of evaluating new quorum systems in this method.

Segment	Node	#Crashes[1]	MTBF[2]	MTTR[3]	MaxTTR[4]	Prob[5]
	hazard	28	157.04	6.98	75.48	4.26%
	havoc	37	119.67	4.46	73.33	3.60%
HU-Lab	hashem	31	142.84	5.31	73.98	3.58%
	harpo	35	124.72	6.50	74.95	4.95%
	hal	15	194.98	111.20	648.77	36.32%
	pita	254	16.92	1.16	51.15	6.40%
HU-Pub	bagel	247	17.74	0.85	48.16	4.57%
	mangal	46	97.96	1.88	47.94	1.89%
	marva	110	41.23	0.52	15.30	1.25%
	al	116	36.59	3.01	186.60	7.59%
WI	kalanit	117	38.19	1.07	68.48	2.72%
	nurit	157	28.08	1.17	66.16	4.01%
	hadar	119	37.59	1.00	30.83	2.60%
	nufar	82	55.19	0.82	21.09	1.46%

[1] Total number of crashes.
[2] Mean time between crashes, in decimal hours.
[3] Mean time to repair (to reboot).
[4] Maximal time to repair.
[5] Fraction of "down time", in percents.

Table 1. Crash distributions.

4. Failure Patterns

4.1. Processor Crashes

The experiment lasted for 4592.8 hours (191 days). The crash distribution is presented in Table 1. The average fraction of "down-time" over all of the machines is 6.08%. The average after discarding the four extreme values (the two highest and the two lowest) is 3.86%. The data clearly shows that relatively high failure probabilities are common.

Three out of the fourteen machines incurred more than 6% down-time, and one even reached 36%.

These values are consistent with the measurements reported by [31] (although the latter were performed using the simple "ping" utility which does not distinguish between real crashes and network partitions). However they are roughly 10 times higher than the crash probabilities reported in [38] for VAXclusters. This can perhaps be attributed to the fact that planned shutdowns are not considered to be crashes in [38], whereas we count any period of time during which the Spread daemon was unable to function as a crash.

A closer look at Table 1 indicates that our machines generally behaved according to one of three patterns:

- The HU-lab machines with the exception of hal had relatively few crashes during the experiment (≈ 30 each) but long times to repair (≈ 5 hours on average).
- The HU-pub machines incurred many crashes (up to 250) but had short times to repair, ($\approx$ one hour on average).
- The WI machines incurred an intermediate number of crashes (≈ 110) with a short time to repair ($\approx$ one hour on average).

The interesting aspect is that the machines within each segment behaved in an homogeneous manner. This can be attributed to the machines' dependence on common resources such as disk servers, name servers, maintenance activities, and power supply. The two exceptions are the machines hal and al. Looking at the MaxTTR value for al shows that roughly 53.5% of its down-time was caused by a single crash that lasted over a week (this happened due to the relocation of the machine). The machine hal is an older model which tends to be neglected for long periods of time.

4.2. Concurrent Crashes

Table 2 shows the distribution of the number of concurrently crashed machines, in comparison to the predicted distribution assuming independent failures based on the individual crash probabilities of Table 1.

The most obvious conclusion that is evident from Table 2 is that crashes are *not* independent. We can see that the measured distribution has a much longer tail than the predicted one, so the crashes are positively correlated. Moreover, the measured probability of the event "0 concurrent crashes" is significantly higher than the prediction, strengthening the above conclusion. The fact that crashes are correlated is consistent with the findings of [38].

An interesting point to note is that we even observed a 12 minute period of time during which *all* 14 machines were down simultaneously, an extremely rare event under the assumption of independence. This event is attributed to a long wide-area power failure that called for a total shutdown. This indicates that even a 50 km geographic distance does not ensure independence.

Concurrent Crashes[1]	Time[2]	Measured Prob[3]	Predicted Prob[4]
0	2241.26	48.80%	38.61%
1	1826.23	39.76%	41.81%
2	261.53	5.69%	15.87%
3	15.28	0.33%	3.25%
4	41.77	0.91%	0.42%
5	29.91	0.65%	0.04%
6	52.60	1.15%	0.00%
7	113.76	2.48%	0.00%
8	6.29	0.14%	0.00%
9	3.66	0.08%	0.00%
10	0.26	0.01%	0.00%
11	0.06	0.00%	0.00%
12	0.00	0.00%	0.00%
13	0.00	0.00%	0.00%
14	0.20	0.00%	0.00%

[1] Number of concurrent crashes.
[2] Total time, out of 4592.80 hours.
[3] Measured fraction of time (in percents).
[4] Predicted fraction of time assuming independent failures, using the individual crash probabilities of Table 1.

Table 2. Concurrent crash distributions.

#Components	Time	Prob
1	4291.61	93.44%
2	272.30	5.93%
3	23.98	0.52%
4	4.71	0.10%
5	0.01	0.00%
6–14	0.00	0.00%

Table 3. Distribution of the number of partitioned components.

4.3. Network Partitions

Table 3 shows that partitions do happen. During over 6.5% of the time the network was partitioned into two or more components. A priori we expected to find partitions with two or three components since our network contains two sites, one of which is composed of two segments. Surprisingly, we observed occasional partitions into four components, and even a momentary five-component partition, which means that partitions occur even within a single segment. Going back to the raw data we found several such occurrences. Looking more closely at the network structure (see Figure 1), we can see that "physical" segments are in fact implemented using several lower level physical devices, allowing for more intricate failure patterns.

Protocols such as the Available Copy protocol [9, 15] rely on never incurring a partition, either for correctness or efficiency. In [33] it is argued that total failures or network partitions do not occur in practice, so the Available Copy protocol is a viable option. Our findings indicate that such protocols may experience partitions even on a single-segment modern network. Thus it is dangerous to ignore the possibility of such events.

	King HU	King WI	Maj All	Maj HU-Lab	Maj HU-Pub	Maj HU	Maj WI	Weight All	Weight WI
hazard	1	0	1	1	0	2	0	3376	0
havoc	0	0	1	1	0	1	0	3566	0
hashem	0	0	1	1	0	1	0	3572	0
harpo	0	0	1	1	0	1	0	3206	0
hal	0	0	1	1	0	1	0	609	0
pita	0	0	1	0	1	1	0	2911	0
bagel	0	0	1	0	1	1	0	3297	0
mangal	0	0	1	0	1	1	0	4282	0
marva	0	1	2	0	0	0	2	4736	4736
al	0	0	1	0	0	0	1	2712	2712
kalanit	0	0	1	0	0	0	1	3879	3879
nurit	0	0	1	0	0	0	1	3445	3445
hadar	0	0	1	0	0	0	1	3929	3929
nufar	0	0	1	0	0	0	1	4565	4565
UnAvail	4.26	1.25	2.97	4.21	3.63	4.16	1.12	2.81	1.12
AvgUnAcc	3.56	2.21	2.52	3.49	3.20	3.45	2.09	2.44	2.09
WorstUnAcc	6.48	5.64	4.44	6.44	5.75	6.39	5.53	4.43	5.52

Table 4. Static vote distributions and results.

5. Quorum System Evaluation

5.1. Static Systems

We have evaluated nine static quorum systems. Table 4 contains the votes used in every system and the results obtained . We selected two centralized solutions, with a monarchy setting. One (King HU) was situated in the HU segment and the other (King WI) in the WI segment. Then we evaluated five majority based systems as follows: One system uses all the machines (Maj All) with a tie breaking vote. This system is predicted to have optimal availability assuming a fully connected network and a uniform failure probability. The four other systems are all confined to a single segment, or to a single geographic site (Maj HU-Lab, Maj HU-Pub, Maj HU, Maj WI). These are possible candidates to be optimal assuming that the network may partition along the segment borders.

Finally, we calculated the aposteriori weights, based on the observed failure probabilities of Table 1, according to formula (1) and scaled up [7]. Using these weights we defined two systems, one using all the machines (Weight All), and the other confined to the WI segment (Weight WI). These systems are predicted to have optimal availability assuming a fully connected network with the non-uniform failure probabilities which we measured.

Looking at Table 4 and Figure 4, we can make the following observations:

- The communication pattern over the Internet in our system is more correctly modeled as a partially connected network. This can be seen from the fact that quorum systems which are confined to one Internet segment have higher availability than those using all the machines (compare the UnAvail columns of Maj All and Maj WI

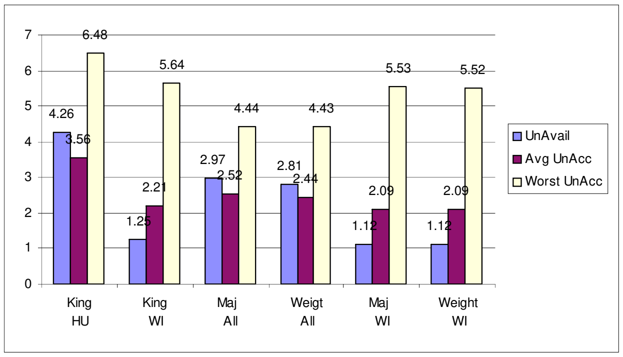

Figure 4. Static quorum systems.

  in Figure 4). This behavior fits the predictions of [8] assuming that each segment is a fully connected sub-net and single links connect the segments (or at least the two geographic sites).

- When the quorum system is confined to a single segment, then using majority on that segment may improve the availability. This can be seen by comparing the UnAvail columns of King WI and Maj WI in Figure 4 and also by comparing the UnAvail data for King HU to the Maj HU, Maj HU-Pub and Maj HU-Lab systems in Table 4. Note that our selection of the kings was somewhat arbitrary: we chose the machines with lowest ID's in each segment. With hind-sight, picking a different king in the HU segment (e.g., hashem) would have achieved a better UnAvail value than majority on this segment. Of course, this information is not available apriori.

- Using the aposteriori weights computed from the observed failure probabilities improved the availability (compare the Maj All to Weight All data and the Maj WI to the Weight WI data), as predicted by [37]. However the improvement is small (less than 0.2%). This can be explained by the fact the computed weights are rather close in value. In fact the smallest quorum in the weighted system contains 6 machines, while in the regular majority the smallest quorum has size 7, so the quorum system defined by these weights is "almost a majority".

- When the quorum system is a monarchy then we found that the worst case un-accessibility value is always poor. However a careful selection of the king's location yields a system with surprisingly good availability (see King WI versus King HU). Using the most highly available processor (marva) as king gave un-availability as low as 1.25%, not much worse than that of the best static systems we checked (Weight WI and Maj WI, both with un-availability of 1.12%). This phenomenon can again be attributed to the segmentation of the network, i.e., the

	Dyn All	Dyn-Weight All	Dyn WI	Dyn-Weight WI	Dyn HU
hazard	1	3376	0	0	1
havoc	1	3566	0	0	1
hashem	1	3572	0	0	1
harpo	1	3206	0	0	1
hal	1	609	0	0	1
pita	1	2911	0	0	1
bagel	1	3297	0	0	1
mangal	1	4282	0	0	1
marva	1	4736	1	4736	0
al	1	2712	1	2712	0
kalanit	1	3879	1	3879	0
nurit	1	3445	1	3445	0
hadar	1	3929	1	3929	0
nufar	1	4565	1	4565	0
UnAvail	2.02	2.11	1.11	1.08	3.01
AvgUnAcc	2.15	2.19	2.08	2.06	2.63
WorstUnAcc	3.98	3.71	5.51	5.49	5.21

Table 5. Dynamic vote distributions and results.

segment where the king resides has good accessibility (and hence good availability and average accessibility) but partitions degrade the accessibility from other segments (and hence the WorstUnAcc value is high).

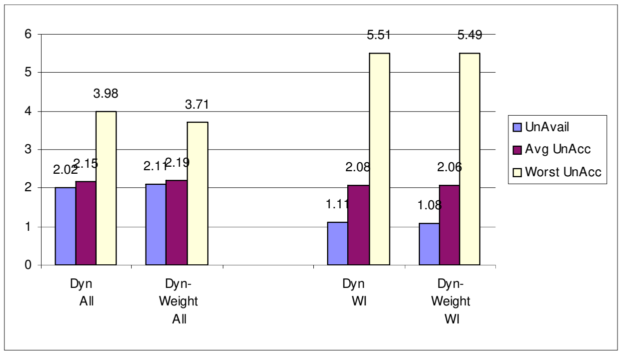

Figure 5. Dynamic quorum systems.

5.2. Dynamic Systems

We have evaluated five dynamic quorum systems, all of which are variants of the Dynamic Linear Voting (DLV) [20]. According to this protocol, when the network configuration changes, e.g., due to a processor failure, a processor recovery, a network partition or a network re-merge, the (single, if any) network component which contains a quorum now defines the entire universe, until the next configuration change. For tie breaking purposes the processors are ordered according to some predefined order.

Table 5 and Figure 5 show the systems that we tested and the obtained results. Three of the systems use the basic DLV protocol, either using all the processors (Dyn All) or confined to one of the sites (Dyn WI, Dyn HU). The two other systems follow the same protocol, however each processor has the weight calculated according to formula (1) and scaled up. Figure 6 presents a comparison between corresponding static and dynamic systems. We have discovered that:

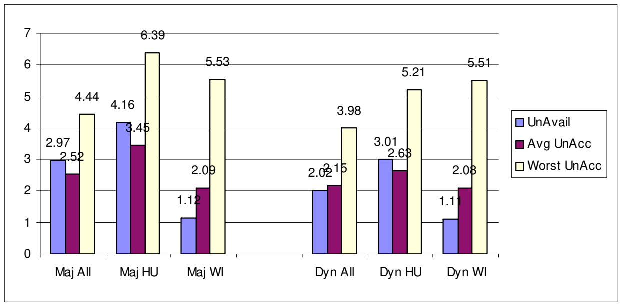

Figure 6. Static versus dynamic quorum systems.

- As in the static case, confining the system to a single segment improves the availability and average accessibility, but also increases the worst case un-accessibility.
- All three quality measures (availability, average and worst accessibility) of dynamic systems were better than that of their static counterparts. This improvement is most evident when partitions are likely to happen (compare Maj All to Dyn All and Maj HU to Dyn HU in Figure 6). However when partitions are rare then the improvement is negligible (i.e., inside a single segment, compare Maj WI to Dyn WI).
- Using the optimal weights had a minor effect in dynamic quorum systems, as was the case for static systems.

6. Clock Drifts

As part of the wide area group communication protocol, each membership change that was observed by the member machines received an agreed identifier (see [27] for details regarding the group communication specifications). This identifier was then written to the local log file of each member machine, along with the local time at the writing machine and the list of current members in the component.

The actual value used as the identifier was the clock time that the *component leader* had when it detected the change (recall Section 3). Therefore in fact every local log record had two time stamps: the local time at the writing machine and its leader's time.

As Table 3 shows, 93.44% of the time our system consisted of a single connected component, and Table 1 shows that hazard (the machine with ID 0) was down 4.26% of the time. Therefore roughly 90% of the time hazard was the only component leader, and the local log files were time-stamped with its clock time in addition to the local clock times. This allowed us to measure the *difference* between each machine's clock and that of hazard. There is inaccuracy in the measured difference since we are ignoring the variations in communication delay and the effects of the protocol, however typically these effects are on the mili-second scale while our clock resolution is in seconds.

In Figure 7 we show the observed difference $T_{\mathrm{hazard}} - T_{\mathrm{other}}$ between hazard's clock and that of four other machines. Each plotted point represents an average taken over a 24-hour period, for each of the 191 days of the experiment.

It is immediately obvious that there is a significant drift between the clocks. The drift has two major factors:

- The effect of the hardware can be seen in the linear slope of the curves. For instance, consider the difference between hazard and harpo's clocks, between day 19 (when the difference was 41 seconds) and day 155 (when it was -180). Clearly harpo's hardware clock rate is faster than that of hazard by a constant factor, and this linear drift is by 1.6 seconds a day, i.e., a linear drift of 18.5 parts in a million. Note that both machines are made by the same manufacturer, run the same operating system, and are situated a few meters apart in the same room.

- The human factor can be seen in the discontinuities in the curves. The "spikes" in harpo and bagel's clocks around day 20 were caused by an operator setting the local clocks to a value wrong by about one hour, and then fixing his error two days later. Note that the correction caused the clocks of harpo and bagel to go over previously counted time values *a second time*. The concurrent jump in all four curves on day 155 was caused by a reset of hazard's clock of about 100 seconds forward. The jitter in the curves during the last month is due to the fact the the HU-Lab machines, including hazard, were moved around and re-configured in this period due to the installation of new equipment, so the clocks were reset several times.

7. Conclusions

We have demonstrated that machine crashes are correlated, that network partitions are frequent (and occur even within a single Internet segment), and that a total system crash is a rare but possible event. As a side effect we have shown that machine clocks manifest significant drift and occasional erratic behavior. All these findings are relevant when analyzing or simulating the performance of a distributed protocol. However we argue that it is especially important not to overlook these phenomena when *designing* a new protocol. A protocol which inherently relies on the assumption that such events never happen could easily reach an incorrect state if used on a system such as ours.

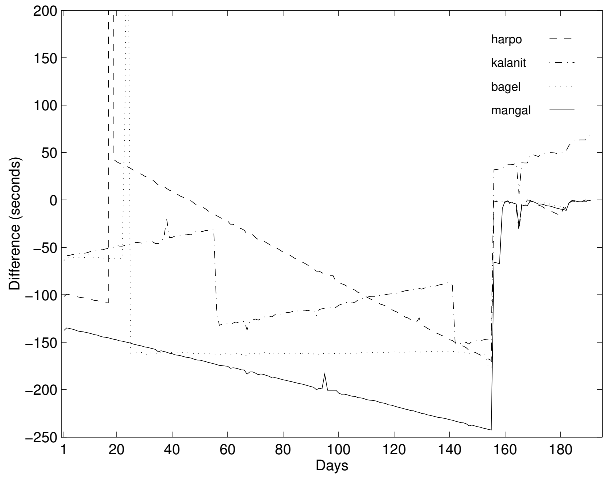

Figure 7. Clock drift relative to Hazard's clock, over time.

As for quorum system performance, we observed that our Internet-based communication network is more accurately modeled by a partially connected network. Each segment can be reasonably modeled by a fully connected network *internally*, however assuming only a single link between the two geographic sites in our system gave better predictions. Quorum systems which are confined to a single segment manifest higher availability than those which are predicted to have optimal availability under the assumption of full connectivity. However the worst case accessibility deteriorates for single-segment quorum systems, since machines in other segments had no access when a partition occurs.

We also found that using a dynamic quorum system improves the service level according to all three quality measures. To a lesser extent, so does using the optimal static weights (based on the aposteriori failure probabilities). However these improvements are negligible when the system is confined to a single segment.

Acknowledgment

We are grateful to Ken Birman for encouraging us to make this experiment, to our advisors David Peleg and Danny Dolev for their support and valuable advice, and to Ray Strong for stimulating discussions regarding clock drifts.

References

[1] D. Agrawal and A. El-Abbadi. An efficient and fault-tolerant solution for distributed mutual exclusion. *ACM*

Trans. Comp. Sys., 9(1):1–20, 1991.

[2] Y. Amir, 1995. Personal communication.

[3] Y. Amir. *Replication Using Group Communication Over a Dynamic Network.* PhD thesis, Institute of Computer Science, The Hebrew University of Jerusalem, Israel. Also available at `http://www.cs.jhu.edu/yairamir`, 1995.

[4] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Membership algorithms for multicast communication groups. In *Proc. 6th Inter. Workshop on Dist. Algorithms (WDAG), LNCS 647*, pages 292–312. Springer-Verlag, 1992.

[5] Y. Amir, D. Dolev, S. Kramer, and D. Malki. Transis: A communication subsystem for high availability. In *Proc. 22nd IEEE Symp. Fault-Tolerant Computing (FTCS)*, pages 76–84, 1992.

[6] Y. Amir, L. E. Moser, P. M. Melliar-Smith, D. A. Agarwal, and P. Ciarfella. The Totem single-ring ordering and membership protocol. *ACM Trans. Comp. Sys.*, 13(4), 1995.

[7] Y. Amir and A. Wool. Optimal availability quorum systems: Theory and practice. *Inform. Process. Lett.*, 65:223–228, 1998.

[8] D. Barbara and H. Garcia-Molina. The reliability of vote mechanisms. *IEEE Trans. Comput.*, C-36:1197–1208, Oct. 1987.

[9] P. A. Bernstein and N. Goodman. An algorithm for concurrency control and recovery for replicated databases. *ACM Transactions on Database Systems*, 9(4), Dec. 1984.

[10] S. Y. Cheung, M. H. Ammar, and M. Ahamad. The grid protocol: A high performance scheme for maintaining replicated data. In *Proc. 6th IEEE Int. Conf. Data Engineering*, pages 438–445, 1990.

[11] S. B. Davidson, H. Garcia-Molina, and D. Skeen. Consistency in partitioned networks. *ACM Computing Surveys*, 17(3):341–370, 1985.

[12] K. Diks, E. Kranakis, D. Krizanc, B. Mans, and A. Pelc. Optimal coteries and voting schemes. *Inform. Process. Lett.*, 51:1–6, 1994.

[13] H. Garcia-Molina and D. Barbara. How to assign votes in a distributed system. *J. ACM*, 32(4):841–860, 1985.

[14] R. Golding and D. D. E. Long. Accessing replicated data in a large scale distributed system. *Int. J. Comp. Simulation*, 1(4):347–372, 1991.

[15] N. Goodman, D. Skeen, A. Chan, U. Dayal, S. Fox, and D. Ries. A recovery algorithm for a distributed database system. In *Proc. 2nd ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, Atlanta, 1983.

[16] M. Herlihy. Dynamic quorum adjustment for partitioned data. *ACM Trans. Database Sys.*, 12(2):170–194, 1987.

[17] M. P. Herlihy. *Replication Methods for Abstract Data Types*. PhD thesis, Massachusetts Institute of Technology, MIT/LCS/TR-319, 1984.

[18] R. Holzman, Y. Marcus, and D. Peleg. Load balancing in quorum systems. In *Proc. 4th Workshop on Algorithms and Data Structures*, pages 38–49, Kingston, Ont., Canada, 1995.

[19] T. Ibaraki, H. Nagamochi, and T. Kameda. Optimal coteries for rings and related networks. *Distributed Computing*, 8:191–201, 1995.

[20] S. Jajodia and D. Mutchler. Dynamic voting algorithms for maintaining the consistency of a replicated database. *ACM Trans. Database Sys.*, 15(2):230–280, 1990.

[21] D. B. Johnson and L. Raab. Effects of replication on data availability. *Int. J. Comp. Simulation*, 1(4):373–392, 1991.

[22] D. B. Johnson and L. Raab. A tight upper bound on the benefits of replica control protocols. *J. Computer System Sci.*, 51:168–176, 1995.

[23] I. Keidar. A highly available paradigm for consistent object replication. Master's thesis, Institute of Computer Science, The Hebrew University of Jerusalem, Israel, 1994.

[24] M. L. Liu, D. Agrawal, and A. El-Abbadi. On the implementation of the quorum consensus protocol. In *Proc. Parallel and Distributed Computing Systems.*, Orlando, 1995.

[25] M. Maekawa. A $\sqrt{n}$ algorithm for mutual exclusion in decentralized systems. *ACM Trans. Comp. Sys.*, 3(2):145–159, 1985.

[26] D. A. Menascé, Y. Yesha, and K. Kalpakis. On a unified framework for the evaluation of distributed quorum attainment protocols. *IEEE Trans. Software Eng.*, 20(11):868–884, 1994.

[27] L. E. Moser, Y. Amir, P. M. Melliar-Smith, and D. A. Agarwal. Extended virtual synchrony. In *Proc. 14th Inter. Conf. Distributed Computing Systems*, pages 56–65. IEEE Computer Society Press, June 1994.

[28] S. J. Mullender and P. M. B. Vitányi. Distributed matchmaking. *Algorithmica*, 3:367–391, 1988.

[29] M. Naor and A. Wool. Access control and signatures via quorum secret sharing. *IEEE Trans. Parallel and Distributed Sys.*, 9(9), Sept. 1998. Extended abstract in Proc. 3rd ACM Conf. Comp. and Comm. Security, 1996.

[30] M. Naor and A. Wool. The load, capacity and availability of quorum systems. *SIAM J. Computing*, 27(2):423–447, Apr. 1998.

[31] M. Ogg, 1995. Personal communication.

[32] C. H. Papadimitriou and M. Sideri. Optimal coteries. In *Proc. 10th ACM Symp. Princip. Distributed Computing (PODC)*, pages 75–80, 1991.

[33] J.-F. Pâris and D. D. E. Long. The performance of available copy protocols for the management of replicated data. *Performance Evaluation*, 11:9–30, 1990.

[34] D. Peleg and A. Wool. The availability of quorum systems. *Information and Computation*, 123(2):210–223, 1995.

[35] D. Peleg and A. Wool. Crumbling walls: A class of practical and efficient quorum systems. *Distributed Computing*, 10(2):87–98, 1997.

[36] M. Raynal. *Algorithms for Mutual Exclusion.* MIT press, 1986.

[37] M. Spasojevic and P. Berman. Voting as the optimal static pessimistic scheme for managing replicated data. *IEEE Trans. Parallel and Distributed Sys.*, 5(1):64–73, 1994.

[38] D. Tang and R. K. Iyer. Analysis and modeling of correlated failures in multicomputer systems. *IEEE Trans. Comput.*, 41(5):567–577, 1992.

[39] T. W. Yan and H. Garcia-Molina. Distributed selective dissemination of information. In *Proc. 3rd Inter. Conf. Par. Dist. Info. Sys.*, pages 89–98, 1994.