Literature Survey Report
On
Quorum Systems

Stefan Witt
WSU ID 10539995
stefan_witt@wsu.edu

CptS/EE 562
March 7, 2002

Abstract

This report summarizes the state of the art in the research area of quorum systems as represented by the papers [PM01], [ES00], [MRW97], and [AMR+00]. Quorum systems are used in fault tolerant distributed computing systems for ensuring the availability of a replicated service and replicated data, even if the replicated servers fail by crashing benignly or in a byzantine, i.e. arbitrary way. Quorums are subsets of the servers with the condition that every pair of subsets intersect, so that each quorum can make a decision on behalf of all servers.

In the papers new quorum systems with improved quorum size, load and availability are introduced and their properties mathematically proved. There are lower bounds for both load and availability, and there are quorum systems that are optimal in respect to these properties.

Two of the papers deal with the emulation of shared memory that can be accessed by servers in a distributed system. Quorums are used to grant read or write access to the shared memory. The problems that arise are how to reconfigure the quorum system in case of a failure, and how to tolerate byzantine failures.

Two papers are about byzantine fault tolerance, i.e. tolerating value faults. Quorum systems that tolerate byzantine failures must suffice stronger requirements for the intersection properties.

The summarized papers are analysed and compared, so that a complete picture of the research area of quorum systems is provided.

1 Overview of Technical Issues

This section introduces the basic definitions for quorum systems that are used in the papers, following [MR98]. The study of quorum systems assumes a set of servers U, which contains n servers, i.e. $|U| = n$. A *quorum system* $\mathcal{Q}$ is defined as a non-empty set $\mathcal{Q} \subseteq 2^U$, every pair of which intersect. Each $Q \in \mathcal{Q}$ is called a *quorum*. That means in a quorum system the servers are organized in overlapping quorums so that each quorum overlaps with every other quorum. Therefore, a decision or an action of one quorum is known in every other quorum, since the quorum intersects with every other quorum. So, a quorum can act on behalf of the whole set of servers.

For tolerating byzantine failures, the intersection property must be stricter. In order to be resilient to b byzantine failures, the quorum intersections must be greater than $2b + 1$. This can be seen as follows. Assume in a quorum intersection are b faulty servers. In order for the system to be correct despite of these faulty servers, it is necessary that the majority of the servers in the intersection is still correct, i.e. there must be at least $b + 1$ correct servers in the intersection. This gives a minimum intersection size of $2b + 1$ servers. Formally, this is defined as follows (cited from [MRW97]): A quorum system $\mathcal{Q}$ is a *b-masking quorum system*, if it is resilient to $f \geq b$ failures, and obeys the following consistency requirement: $\forall Q_1, Q_2 \in \mathcal{Q} : |Q_1 \cap Q_2| \geq 2b + 1$.

Characteristics of quorum systems are quorum size, load, and availability. The quorum size is simply the number of servers in each quorum; it is assumed that each quorum has the same size.

To define the load, some other definitions are needed. An *access strategy* w is a probability distribution on the elements of the quorum system $\mathcal{Q}$, i.e. it is normalized to $\sum_{Q \in \mathcal{Q}} w(Q) = 1$. $w(Q)$ is the probability that quorum Q will be chosen when the service is accessed. The *load induced by w on a server u* is defined as $l_w(u) := \sum_{Q \in \mathcal{Q} \text{ s.t. } u \in Q} w(Q)$, i.e. it is the probability that any quorum is chosen that u is member of. Furthermore, the *load induced by a strategy w on a quorum system* is $\mathcal{L}_w(\mathcal{Q}) := \max_{u \in U}\{l_w(u)\}$, i.e. this is the maximum load that is put on a server if the strategy w is used. Finally, the *system load* (or just *load*) on a quorum system $\mathcal{Q}$ is defined as follows: It is $\mathcal{L}(\mathcal{Q}) := \min_w\{\mathcal{L}_w(\mathcal{Q})\}$, i.e. it is the minimum of all possible loads induced by strategies w. Therefore, the load of a quorum system is achieved only if an optimal access strategy is used and if no failures occur. If failures occur, then the quorum system must be reconfigured.

The definition of the availability is taken from [MRW97]. To measure the availability of a quorum system, the crash probability is used. It indicates the probability that for an equally distributed failure probability p of all servers at least one quorum system survives with no crashed, i.e. faulty, members. Formally, it is defined as follows (cited from [MRW97]). Assume that each server in the system crashes independently with probability p. For every quorum $Q \in \mathcal{Q}$ let ϵ_Q be the event that Q is hit, i.e. at least one element $i \in Q$ has crashed. Let $crash(Q)$ be the event that all the quorums $Q \in \mathcal{Q}$ were hit, i.e. $crash(Q) := \bigwedge_{Q \in \mathcal{Q}} \epsilon_Q$. Then the system crash probability is $F_p := \mathbb{P}(crash(\mathcal{Q}))$.

2 Paper Summaries

2.1 Revisiting Hierarchical Quorum Systems

In this section, the paper "Revisiting Hierarchical Quorum Systems" [PM01] by N. Preguiça and J. Martins is presented and discussed. The paper introduces and analyses two quorum systems, the hierarchical T-grid, and the hierarchical triangle quorum system. Both of them will now be briefly described, following [PM01].

Hierarchical T-grid

A hierarchical T-grid is a quorum that can be used to grant either mutually exclusive read/write or read access to a client. All servers are ordered in a grid shape, and they are combined to logical objects. Logical objects again can be combined to a higher order logical object, and therefore, a hierarchy of server groups is formed, as shown in figure 1: It shows a two-level grid with 16 servers, where 2×2 groups are combined to a logical object.

A hierarchical T-grid quorum is formed by a full-line cover and a partial row cover in respect to the full-line cover in the top level of the hierarchy. A *full-line cover* in a level i grid object is formed (recursively) obtaining a full-line in all objects of at least one row of the corresponding level $i-1$ grid. A full-line in a level 0 object is defined as the object itself. A *row-cover* in a level i object is formed (recursively) obtaining a row-cover in at least one object of every row of the corresponding level $i-1$ grid. A row-cover in a level 0 object is defined as the object itself. A *partial row-cover* in respect to a given set S is formed by removing all those objects from a row-cover, that are above a topmost element of S.

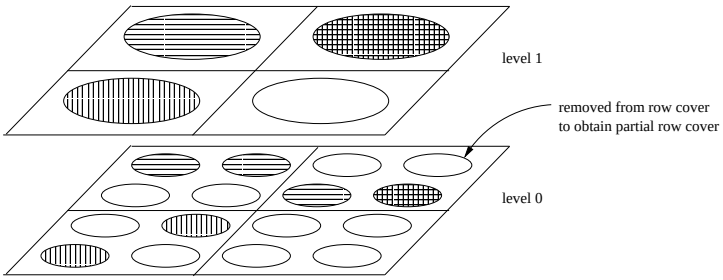

Figure 1: Hierarchical T-grid with $i = 2$ levels; level 2 is not shown. A quorum is marked: Full-lines are marked with horizontal lines, and partial row-covers are marked with vertical lines. Figure taken from [PM01] and modified.

In the example of figure 1, each logical object at level 1 consists of 4 servers at level 0. A quorum is shown: In level one there are a full-line cover (back left and back right objects) and a row cover (front left and back right objects), which form a quorum. The covers in level 1 are obtained as follows: The back left object represents a full-line cover in level 0, and the front left object represents a row-cover in level 0. The back right object in level 1 represents a full-line in level 0 and a partial row-cover. The row-cover is partial, because the back right server at level 0 (in the back right square) has been removed, because it it above the full-line.

The hierarchical T-grid quorum is analysed in [PM01]. The failure probability has been analysed empirically. The result is that the hierarchical T-grid has a 7.5% - 10% lower failure probability than the hierarchical grid. Furthermore, the failure probability decreases asymptotically to 0 for a larger number: $\lim_{n\to\infty} F_p(\text{h-T-grid}) = 0$. The quorum size varies depending whether quadratic or rectangular grids are used: $\sqrt{n} \leq |\text{quorum}| \leq 2\sqrt{n} - 1$. The load of the system has been shown to be $\mathcal{L}(\text{h-T-grid}) \geq \frac{3}{2}\sqrt{n}$.

Hierarchical Triangle

The hierarchical triangle is also a recursively organized quorum system. The servers are combined in a triangle with i rows, where the i^{th} row contains i servers. A triangle of level m is recursively composed out of 3 parts: A sub-triangle of the first $\lfloor \frac{i}{2} \rfloor$ rows, a sub-grid of the first $\lfloor \frac{i}{2} \rfloor$ servers of rows $\lfloor \frac{i}{2} \rfloor + 1$ to i, and another sub-triangle consisting of the remaining servers. For $i = 5$ rows the hierarchical triangle is shown in figure 2. A hierarchical triangle quorum is formed by obtaining a quorum at level 0 in the triangle, and a quorum at level m in a triangle can be achieved by one of the following rules:

- A triangle that consists of a single server is a quorum.
- If the triangle has more than one server, then a quorum can be obtained by one of the following rules (T_1 and T_2 are the sub-triangles, and G is the sub-grid):
  1. If A is a quorum in T_1 and B is a quorum in T_2, then $A \cup B$ is a quorum.
  2. If A is a quorum in T_1 and B is a row-cover in G, then $A \cup B$ is a quorum.
  3. If A is a quorum in T_2 and B is a full-line in G, then $A \cup B$ is a quorum.

The properties of the hierarchical T-grid quorums have been analysed in [PM01]. The failure probability has been examined empirically. It is better than all quorum systems that are cited in the paper. For a large number of servers, the failure probability decreases. The quorum size is constant, and the load is $\mathcal{L} = \sqrt{\frac{2}{n}}$ and therefore almost optimal.

Results

The main contribution, which this paper has made, is the introduction of two quorum systems with improved properties compared to the references it cites. The hierarchical grid has been modified to the hierarchical T-grid, which has a smaller average quorum size and a smaller failure probalility. The second introduced quorum system has almost optimal load. Both quorum systems have a smaller failure probability than all previously known hierarchical quorum systems.

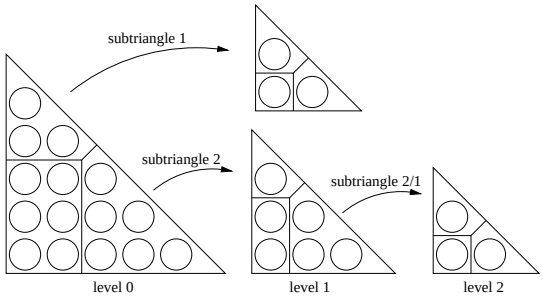

Figure 2: Hierarchical Triangle quorum with i=5 rows and columns. The quorum decomposition into subtriangles is illustrated. Figure taken from [PM01] and modified.

Strengths

Strengths of this paper are the formal mathematical proofs of correctness of the quorum systems that are presented. Along with the comparison with the other hierarchical quorum systems, the paper is convincing that the proposed hierarchical quorum systems are better than the previously known ones.

Limitations

However, the quorum systems that were presented, have some limitations. They do not cover byzantine failures, and they do not allow reconfiguration of the quorums. This is a fundamental disadvantage of these quorum systems, because they cannot be modified to be resilient to byzantine failures. Both quorum systems that have been proposed do not have the required property that the quorum intersections have the size of at least $2b + 1$ servers, where b is the number of tolerated byzantine faults, and this property cannot be added to the systems easily. If the hierarchical quorum systems are used in the context of a reconfiguration service, as proposed in [ES00], however, a reconfiguration can be done, but still a method for doing this is missing. The paper assumes that the quorum system is fixed in the design phase of the distributed system.

Another problem is that the failure probabilities have been obtained empirically. So it is not possible to analyse a particular quorum system using some formula. The provided analysis tables are quite limited, because they show results only for some specific small n. In order to analyse hierarchical quorum systems before deploying them in a distributed system, an analytical formula is needed to calculate the load and crash probability of the system.

2.2 Graceful Quorum Reconfiguration in a Robust Emulation of Shared Memory

In this section the paper "Graceful Quorum Reconfiguration in a Robust Emulation of Shared Memory" [ES00] by B. Englert and A. Shvartsman is summarized.

A method is presented how to implement memory that is shared between distributed servers. All servers can read or write the memory, i.e. there can be multiple readers and writers. Each server has a local copy of the memory, and coordination is accomplished by exchanging messages. Reading and writing quorums are used for coordination. For fault tolerance, the quorum system can be reconfigured by a dedicated server, the reconfigurer. A reconfiguration is done when servers fail and a monitor server requests to install a new quorum system.

The algorithm for simulating a shared memory uses the Γ primitive that is introduced in [LS97] as follows. Γ provides a quorum-acknowledged broadcast and allows changing quorum configurations. Its interface is shown in figure 3. It is invoked with the *submit* call, whose parameters are the message m, the voting function (condenser function) ψ, a quorum $Q \in \mathcal{Q}$, and a unique *message id*. The message is broadcasted to all servers via the *deliver* call, which takes the message m and its id as arguments.

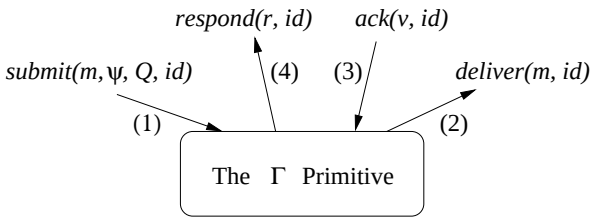

Figure 3: Interface of the Γ primitive. Figure taken from [LS97].

Then the Γ primitive collects all broadcast responses from the servers. The responses come in via the *ack* call; they contain a return value v and the *message id*. When all servers have answered, the voting function ψ is applied to those return values that have been sent by the specified quorum Q. The voting result r along with the message id is sent back to the caller with the *respond* call.

In particular the algorithm for shared memory emulation works as follows: Each server holds a copy of the shared memory represented as a single value val, and a tag that consists of a sequence number and a server identifier. The tag represents a version number of the memory. Furthermore, each server p has a pair of quorum configurations $cfg_p = (cfg.act_p, cfg.bid_p)$ and quorum indices $cix_p = (cix.act_p, cix.bid_p)$. The indices are the sequence numbers of the active configuration $cix.act_p$ and the number of the proposed configuration $cix.bid_p$. $cfg.act_p$ and $cfg.bid_p$ are the active and proposed configuration, respectively. The read and write operations are performed in 2 phases:

1. *Query phase*: The server that wants to read or write uses the Γ primitive to query all servers in a read quorum for their tags.

   The query response consists of the values, tags, configuration index pairs, and configuration pairs of all servers. The server determines the maximum configuration index and checks if it is greater than the index used for querying. If this is the case, then the server has used an obsolete

read quorum of an old quorum configuration. It handles this situation as follows: It performs a *quorum-join* by merging the used quorum and the quorum of the newest configuration and thus obtaining an *intermediate quorum*. It sets its proposed configuration to the newest configuration and starts over the query phase using the newest read quorum.

When the query respond that consist of the server tags comes in, the server constructs a propagate tag $prop\text{-}tag$, and a propagate value $prop\text{-}val$. The $prop\text{-}tag$ is constructed like this: The server identifier of this server is used, and a new sequence number is determined. If this server is a writer, then it is set to the maximum sequence number of all servers plus 1, thus it becomes the highest sequence number of all. If this server is a reader, then the propagated sequence number becomes the maximum sequence number of all queried servers. The $prop\text{-}val$ is set to the value of the tag with the maximum sequence number; a writing server does its write operation on $prop\text{-}val$.

2. *Propagate phase*: The server uses the Γ primitive to propagate $prop\text{-}tag$ and $prop\text{-}val$ to all servers in a write quorum.

There is also one reconfigurer in the distributed system that is designated to establish a new quorum configuration, if this is necessary due to server failures. The reconfigurer also maintains a pair of quorum configurations $cfg_r = (cfg.act_r, cfg.bid_r)$ and quorum indices $cix_r = (cix.act_r, cix.bid_r)$. The configuration index cix_p at any server p is defined to be *current*, if $cix_p \geq cix_r$. The reconfigurer works in 3 phases:

1. *Query-install phase*: The reconfigurer informs the joined active read quorum and active write quorum about the proposed new configuration and queries the values and tags from it.

2. *Propagate phase*: The reconfigurer propagates the maximum tag and the associated value to a write quorum in the new configuration.

3. *Recon-idle phase*: The reconfigurer announces to to a write quorum in the new configuration that the reconfiguration is complete. By doing this, it informs the servers that from now on only the new configuration is used.

Results

The results of the introduced shared memory emulation extend those from [LS97]. The most important improvement is that a fault-tolerant reconfiguration service has been added. In [LS97] only client servers were tolerated to fail, whereas [ES00] also tolerates the reconfigurer to fail, i.e. the reconfigurer has been eliminated as a single point of failure. The basic idea to accomplish this fault tolerance is to use intermediate quorum configurations, which consist of the old and the new quorum configuration. If the reconfigurer fails during reconfiguration, the system still works; the proof for correctness and safety is sketched in [ES00].

A property of the use of intermediate quorum configurations is that during reconfiguration readers and writers do not have to wait for completion of the reconfiguration. They simply query an intermediate quorum in their query phase and obtain a read or write majority.

The analysis of the shared memory emulation resulted in bounded time delays and in bounded number of exchanged messages for both reconfiguration and for access queries. Any read or write operation

that does not fail, takes at most $10g+5dg$ time and $(2d+4)n$ messages (g is the upper bound to perform a computation, d is the distance between the highest configuration and the configuration of this server, and n is the number of servers). Any reconfiguration takes at most $15g$ and $6n$ messages. Therefore, the liveness of the protocol is ensured.

Strengths

The strength of this paper is the flexibility of its results. The protocol that has been introduced can be used as a powerful building block in a distributed system. It provides the abstraction of a shared memory that can be randomly accessed from all servers in the distributed system. This memory is fault tolerant, because it still works if servers fail, and if the reconfigurer fails. The operation is not interrupted when failures occur; even during reconfiguration, the readers and writers do not have to wait.

Furthermore, an implementation is proposed. Although the implementation is abstract in order to prove its correctness, a concrete implementation can be designed from the abstract specification. The architecture is two-layered: It is split into the Γ primitive and the actual memory emulation.

Another strength is the independency from a quorum system. An arbitrary quorum system can be used with this protocol. So it is possible to choose a quorum system with the desired load, availability, and quorum size properties according to the application. Especially, it is possible to use a byzantine quorum system in order to mask byzantine failures of one of the servers. This is done in [AMR+00].

Limitations

There are several limitations and problems in the protocol that is presented in [ES00]. First, the limitations due to the reconfiguration are shown. The protocol does not propose a mechanism to exchange the reconfigurer. The failure of the reconfigurer is concerned, but it is assumed that the reconfigurer recovers and continues its operation. However, if it is not recovered, then no more reconfigurations can be performed, and the distributed system will eventually fail, if the majority of servers of one quorum fail. A mechanism is needed to switch to a new reconfigurer.

Another issue that deals with the reconfigurer is a security problem: The reconfiguration is initiated by calling the reconfigure primitive of the management interface of the reconfigurer. Usually, a monitor station does this, but an intruder could also do this call. A possible attack is to permanently send new configurations to the reconfigurer and thus slowing down or stopping the system. A stop is theoretically possible because the time bound depends on the number of newly established configurations.

Another problem of the reconfigurer is that the protocol assumes that it has no byzantine failures. A byzantine reconfigurer that sends inconsistent information about new quorum configurations will let the whole distributed system go to an inconsistent state. Byzantine failures of the other servers, however, can be tolerated by using a byzantine quorum system.

A limitation on the client side is that at least one quorum is needed in order to make progress. However, this is not a bad property, since the whole distributed system has failed anyway if all quorums have failed.

A more severe problem on the side of the servers is that the sequence numbers that are used to distinguish between different configuration versions may wrap around. The paper defines a lexicographic order on the sequence numbers, which is violated if a wrap-around occurs. In this case, a new con-

figuration cannot be established, because the servers reject the sequence number as old, when they determine the maximum sequence number. The system will fail eventually, if no new quorum system can be established. It has to be considered how often a new configuration is propagated by the reconfigurer and after what time the wrap-around occurs. Reconfigurations do not happen very often, and therefore, a 32 bit configuration index is sufficient in most cases.

An inherent limitation of the protocol is the performance. Each server has a copy of the emulated shared memory in its own memory, and for each read and write operation the whole shared memory is transmitted within the exchanged messages. This has a very high cost of network bandwidth and a high delay. Therefore, the shared memory is limited in size in order to maintain high performance.

2.3 The Load and Availability of Byzantine Quorum Systems

The paper "The Load and Availability of Byzantine Quorum Systems" [MRW97] by D. Malkhi, M. Reiter, and A. Wool is presented in this section.

Four new b-masking quorum systems are introduced in [MRW97], which tolerate b byzantine failures transparently to the servers. General conditions for the existence of masking quorum systems are $4b < n$, where n is the number of servers, and quorum intersections of the size $2b + 1$. This minimum intersection size is necessary to mask byzantine quorum systems, because it means that in any quorum intersection there is still the majority of servers correct.

The M-Grid Quorum System

The first quorum system that is introduced in [MRW97] is the M-grid system. The n servers are arranged in a grid of $\sqrt{n} \times \sqrt{n}$. A quorum consists of exactly $\sqrt{b+1}$ rows and $\sqrt{b+1}$ columns of the grid. This is shown in figure 4 for $n = 49$ servers and $b = 3$ masked byzantine failures, where one quorum is marked. The intersection property is fulfilled, because different quorums intersect at more than $2b+1$ servers, which is shown in [MRW97]. Furthermore, it has been shown that M-grid is a b-masking quorum system.

Analysis of the M-grid system resulted in an optimal load of $\mathcal{L}(\text{M-grid}(b)) = \sqrt{\frac{2b+1}{n}}$ for $b \approx \frac{\sqrt{n}}{2}$, and the crash probability converges to 1, $\lim_{n\to\infty} F_p(\text{M-grid}) = 1$. So, for a constant crash probability of servers the system will crash, because there is at least one crash in each row or column.

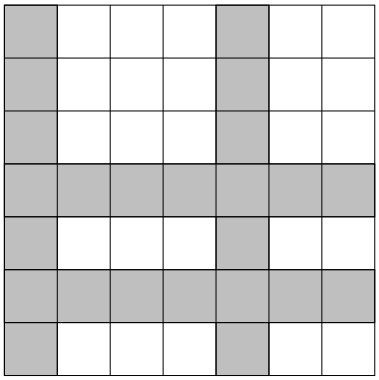

Figure 4: M-grid quorum system with $\sqrt{n} = 7$. Figure taken from [MRW97].

RT Quorum System

Another quorum system defined in [MRW97] is the RT system. It is constructed by taking l-of-k threshold systems (with $k > l > \frac{k}{2}$) and arranging them in a k-nary tree of depth h, as shown in figure 5. A quorum is defined recursively. A quorum is formed if l out of k results at the root node of the tree are the same. This majority of l out of k must hold at each inner node of the tree. If the

subnodes of a threshold system are leaf nodes, then the threshold system returns the result that is given by l out of k leaves (servers). An example quorum is marked in figure 5.

It has been shown in the paper that a $RT(k,l)$ system is a b-masking quorum system, if the number of tolerated byzantine failures is $b = \min\{\left(n^{\log_k(2l-k)}-1\right)/2, n^{\log_k(k-l+1)}-1\}$. The load and crash probability have also been examined. The load is $\mathcal{L}(RT(k,l)) = \frac{c(RT(k,l))}{n}$, which is not optimal. However, in some cases, optimal load can be achieved. An example is given in the paper.

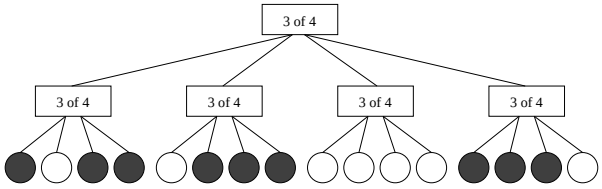

Figure 5: RT quorum system with $k = 4$, $l = 3$, and $h = 2$. One quorum is marked. Figure taken from [MRW97].

The behavior of the crash probability depends on the failure probability p of the servers. If it is smaller than a critical probability p_c, then the failure probability F_p converges to 0, and if $p > p_c$, then F_p converges to 1. The critical probability can be calculated, as shown in the paper, and therefore, an optimal crash probability can be achieved.

boostFPP

The third quorum system that is introduced in [MRW97] is boostFPP. The name indicates that this system is an extended (boosted) version of FPP, i.e. the finite projective plane quorum system. The latter is not introduced in the paper, but rather a reference is given. FPP of order q is a quorum system that consists of $n_F = q^2 + q + 1$ servers and has quorums of size $c(\mathrm{FPP}) = q + 1$. It is regular, i.e. it has intersections of size 1, and therefore, it cannot be used to mask byzantine failures. To accomplish the masking of these, FPP is composed with a threshold system $(3b+1)$-out-of-$(4b+1)$, i.e. $4b+1$ FPP quorum systems are used, and any majority of $3b+1$ of them forms a boostFPP quorum. This quorum system boostFPP(q,b) has $(4b+1)(q^2+q+1)$ servers with quorums of size $(3b+1)(q+1)$.

In [MRW97] it has been proved that boostFPP is a b-masking quorum system. The analysis showed that the load is $\mathcal{L}(\mathrm{boostFPP}(q,b)) \approx \frac{3}{4q}$. This is an important result, because the load is only dependent on the number of FFP quorum systems, but not on the number of tolerated byzantine failures. Therefore, the load is not increased when the number of tolerated byzantine failures is increased, and adding servers to the system, i.e. increasing q, results in a decreased load.

The examination of the failure probability showed 2 facts. First, if the failure probability p of the servers is greater than $\frac{1}{4}$, then the crash probability of the system converges asymptotically to 1, i.e. $\lim_{n\to\infty} F_p(\mathrm{boostFPP}) = 1$. Second, if $p < 0.25$, then there is an upper bound on the crash probability, $\exp\left(-\Omega(b - \log q)\right)$, where Ω is the Landau notation for an upper bound. It can be seen that for a constant q the crash probability becomes asymptotically optimal, otherwise it is suboptimal, yet it does not converge to 1.

The M-Path Quorum System

The last quorum system that is presented in [MRW97] is M-path. The servers are organized in a square grid of the points $\{(i,j) \in \mathbb{Z} : 1 \le i,j \le \sqrt{n}\}$. The grid has an undirected edge between two points (i_1,j_1) and (i_2,j_2) if one of these conditions holds:

1. $i_2 = i_1$ and $j_2 = j_1 + 1$ (up),
2. $i_2 = i_1 + 1$ and $j_2 = j_1$ (right),
3. $i_2 = i_1 - 1$ and $j_2 = j_1 + 1$ (diagonally up and left).

A quorum in M-path is formed by $\sqrt{2b+1}$ disjoint paths along edges from the left side to the right side and $\sqrt{2b+1}$ paths along edges from the top side to the bottom side. An example quorum is illustrated in figure 6.

M-path has been analysed in [MRW97]. It has been proved that it is a b-masking quorum system. M-path has an optimal load: $\mathcal{L}(\text{M-path}(b)) \leq 2\sqrt{\frac{2b+1}{n}}$. It also has an optimal crash probability F_p, because it has been proved that an upper bound of the crash probability is $\exp(-\Omega(\sqrt{n}-\sqrt{b}))$. This bound holds for every server failure probability $p < 0.5$, and it asymptotically converges to 0 for an increased server number, i.e. $\lim_{n\to\infty} F_p(\text{M-path}) = 0$, which is optimal.

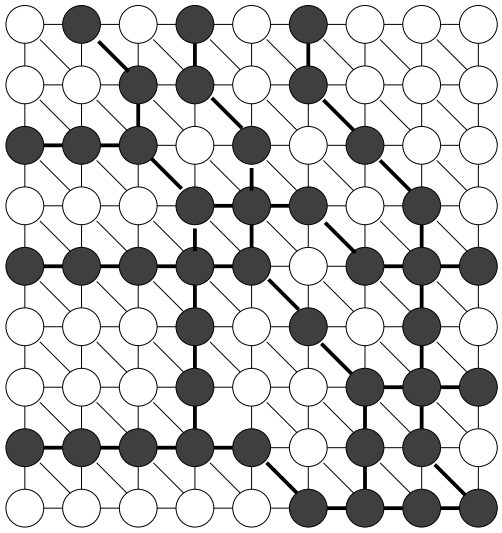

Figure 6: M-path quorum system with $\sqrt{n} = 9$. One quorum is marked. Figure taken from [MRW97].

Results

The main results of this paper are the theoretical considerations of load, and crash probability of b-masking quorum systems. The analysis showed bounds for both measures, a lower bound of the load and a lower bound of the crash probability. Four practical byzantine quorum systems have been presented, each of which is optimal in at least one of the measures. The last one, M-grid, is optimal in both load and crash probability, which is a very important result, because it shows that the theoretical lower bound can actually be reached by a practical quorum system.

Another result of the paper is a method to transform an arbitrary benign fault tolerant quorum system into a byzantine one. This is accomplished using the boosting technique that has been used to introduce the boostFPP quorum system. This method can generally be applied to quorum systems other than FPP.

Strengths

The basic strength of the paper [MRW97] is that its theory is mathematically founded and has been proved. The properties of the presented quorum systems have also been proved, and therefore, they are reliable.

Another strength is that different quorum systems with different properties have been presented. Depending on the application of the quorum system and its requirements, one of these systems can be chosen. An example has been given in the paper, where tradeoffs between load, crash probability, and tolerated byzantine and benign failures are shown.

Limitations

The paper shows quorum systems and proves that they are optimal. However, what is missing is an actual implementation of the systems in order to analyse their practical performance.

2.4 Dynamic Byzantine Quorum Systems

This section presents the paper "Dynamic Byzantine Quorum Systems" [AMR+00] by L. Alvisi, D. Malkhi, E. Pierce, M. Reiter, and R. Wright.

The paper deals with the adaption of the number b of masked byzantine failures in a quorum system. It proposes a method to change this resilience threshold value dynamically in a distributed system. Two different novelties are presented, a method to adjust and read the threshold value, and a shared memory emulation, in which the threshold value can be changed dynamically.

Threshold Adjustment

The current threshold value is held in a shared variable $\mathcal{B}$, which has a value within the range between $b_{\min}$ and $b_{\max}$. Is has a timestamp $\mathcal{T}_{\mathcal{B}}$ to distinguish between different value versions. So, each update is stamped with a unique timestamp that is greater than any previously used one.

For the access of $\mathcal{B}$ a quorum system is needed that is stricter in its definition than b-masking because it has to intersect all possible quorums of size between $b_{\min}$ and $b_{\max}$. Therefore, a *threshold masking quorum system* is defined as a quorum system with quorum sizes of $|Q| = \lceil \frac{n+2b+1}{2} \rceil$. Quorums of a threshold masking quorum system are called *announce sets*, because they are holding the latest value for $\mathcal{B}$ and can be queried for it.

The write of a new threshold value works as follows. Suppose a server wants to set a new threshold value b. It queries servers of an announce set to obtain the values of $\mathcal{B}$ and $\mathcal{T}_{\mathcal{B}}$. It takes the value with the newest timestamp, updates the associated value, and increases the timestamp by 1. Then the client sends the updated $\mathcal{B}$ and $\mathcal{T}_{\mathcal{B}}$ to the announce set. Because the announce set is a majority in each b-masking quorum, the new threshold value is propagated. The actual reconfiguration of the byzantine quorum system has to be done by the announce set as well.

The reading of the threshold value by a client is done similarly. The client queries servers in a threshold masking quorum in order to obtain the current values of $\mathcal{B}$ and $\mathcal{T}_{\mathcal{B}}$. Out of the value-timestamp pairs it selects the pair that has been returned by at least $b_{\max}+1$ servers and that has not been countermanded, i.e. it is not obsoleted by a write with a newer timestamp.

This algorithm has been proved to be correct.

Variables in Dynamic Threshold Systems

The paper also describes an algorithm to share a variable between the servers. To achieve this, the previously introduced algorithm has to be extended. The problem is the increase of b. If a value has been written before the threshold was incremented, then reading from a quorum of the size $\frac{n+2b+1}{2}$ is not enough to ensure $b+1$ correct responses, because some servers might still respond with the old value. Therefore, the quorum size used for the reading is increased by $b - b_{\min}$.

The writing is also complicated by an increased threshold b, because the writer needs to obtain the most recently written value of $\mathcal{B}$. If the quorum that the writer accesses has only obsolete, i.e. countermanded, values of the threshold, then the writer has to query a larger quorum of the maximum size $\frac{n+2b_{\max}}{2}$ in order to obtain the latest value.

In [AMR+00] there is a protocol for the reader and writer for sharing a common variable.

Results

The main result in the paper [AMR+00] is the method to adapt the number of tolerated byzantine failures in a quorum systems at run time. Therefore, the number of tolerated byzantine failures of servers in a distributed system does not have to be decided in the design phase any more. This is advantageous, because no resources are wasted in order to maintain resilience to a high number of byzantine failures. The system can work very efficiently when the extra resources to mask byzantine failures are not used all the time.

The second result is the algorithm to share a variable between multiple readers and writers, while the dynamic change of the threshold value is still possible. So, the shared memory emulation that has been introduced in [ES00] is extended to mask byzantine failures.

Strengths

An advantage of the method to adapt the number of masked byzantine failures dynamically is that strong semantics is provided. That means, the next read after a write of a new value for $\mathcal{B}$ results in the new value. Of course, this is an implication of the correctness of the algorithm, but it is important to notice.

Furthermore, the algorithm can be used to implement shared memory between multiple servers, where the memory access tolerates byzantine failures. This is simply done by emulating the memory as a number of shared variables.

Another advantage is the independence from specific quorum systems. In the paper it has been shown that the adaption works with boostFPP [MRW97] and M-grid [MRW97] as well. It can be used with any arbitrary byzantine quorum system, and thus it has the ability to be scaled in respect to load and crash probability depending on the requirements of the application.

Limitations

A limitation of the proposed method for adapting the number of masked byzantine failures is the assumption that clients and channels are reliable. In practice, this is not always the case. Links and clients can fail as well, and clients can even fail in a byzantine way, for instance in case of an intruder.

Like in [ES00], this method does not deal with the problem of a possible wrap-around of sequence numbers. In this case the version number of $\mathcal{B}$ might wrap around, and the algorithm will fail eventually, because a new threshold value cannot be established.

What is missing in this paper is an analysis of the time that is needed to establish a new threshold. It is shown that the algorithm is correct, but a time limit is required for some applications, especially for distributed real-time systems.

3 Analysis

The four papers that have been summarized in section 2 are all about quorum systems, but they deal with different issues in this field of research. They can be categorized in two main areas. The first one is the introduction and theoretical analysis of quorum systems. The papers [MRW97] and [PM01] fall into this category. The second field of study is the application of quorum systems for emulation of shared memory. The two other summarized papers, [AMR+00] and [ES00], deal with this. They do not make assumptions about the underlying quorum system, and therefore, they are usable with any possible quorum system. However, the method to change the threshold for byzantine quorum systems proposed in [AMR+00] only makes sense when byzantine quorum systems are used; but still an arbitrary byzantine quorum system can be used.

Introduced Quorum Systems

The first category of papers is analysed now. The paper [MRW97] introduces four byzantine quorum systems and theoretically analyses their load and crash probability (see section 2.3). In contrast to that the paper [PM01] introduces two hierarchical quorum systems and analyses them empirically, i.e. determines values of load and availability by induction. This is a weakness of the latter paper, because no general statements can be made about the performance of the quorum systems when the server crash probability and the number of servers are given. For a proper system design an analytical formula is needed in order to determine the best quorum system configuration.

A comparison of the results of the quorum systems, which are introduced, shows that the quorum systems of [MRW97] are generally better than those presented in [PM01]. They are optimal in load and/or crash probability, whereas the hierarchical quorum systems both are only *almost* optimal. This only a minimal difference, but is shows that hierarchical quorum systems are inherently worse in their probabilities than non-hierarchical quorum systems.

In general it can be said that hierarchical quorum systems are worse than non-hierarchical quorum systems in respect to the failure probability. A failure of a server in a subcomponent of a hierarchical quorum system is a fault in the supercomponent that uses the subcomponent. Therefore, failures of a server affect the whole hierarchy instead of just the quorums it is part of.

The hierarchical quorum systems that are introduced in [PM01] do not tolerate byzantine failures, because the size of the intersection of the quorums cannot be set as a parameter. The intersection size is fixed to 1, and the shown hierarchical quorum systems cannot be easily modified to tolerate byzantine failures. The hierarchical T-grid always has quorum intersections of size 1: Partial row covers can be obtained in any columns, so there are always two quorums that differ by exchanging two columns in a row cover, i.e. they differ in exactly one server. The hierarchical Triangle always has intersections of at least size 1 at the highest level. Consider for instance the triangle of level 2 in figure 2 (page 4). Two quorums that differ in exactly one level 2 triangle must have an intersection size of 1, since they must have one of the three servers in the triangle in common according to the definition. Therefore, neither of the introduced hierarchical quorum systems tolerates byzantine failures.

Shared Memory Emulations

The second category of papers is analysed now. In [ES00] an algorithm is presented to emulate shared

memory in a distrubted system that uses read and write quorums for accessing it. The algorithm allows multiple readers and writers as well as reconfiguration of the read and write quorums at run time. The paper [AMR+00], however, is better than [ES00], because it develops the algorithm further. It uses a byzantine quorum system with the shared memory emulation algorithm, and furthermore, it allows a dynamic change of the number of tolerated byzantine failures at run time.

Originally, the algorithms that are proposed in both papers are introduced for usage of a shared variable. By using many shared variables or by using an array variable, shared memory can be emulated. However, there is a huge communication overhead in order to use a shared memory emulation. For each memory access, the whole memory is transmitted twice over the network. The performance cost of network bandwidth and local memory accesses is high, and also the local memory cost, because each server holds a local copy of the shared memory.

An alternative solution that has a higher performance uses active replication in order to emulate shared memory. Each server that wants to share the memory has an active local replica. Each time a memory access is done a value is read from the local copy or written to the local copy and sent to the other replicas using a totally ordered multicast. Each server completes the memory accesses deterministically, and therefore maintains the same state as the other replicas. This solution is more performant than the algorithms of the cited papers, because it does not transmit the whole memory content each time an access to the shared memory is requested. It can also be tolerant to byzantine failures. To accomplish this, a byzantine quorum system can be used to agree on the values that are read from the shared memory. A dynamic change of the number of tolerated byzantine failures, however, can be done using the method proposed in [AMR+00].

4 Conclusion

In this report the four papers [PM01], [ES00], [MRW97], and [AMR+00] have been summarized, analyzed and compared. These papers are the current state of the art in quorum systems, which are used in fault tolerant distributed computing systems.

The papers' contribution to the research area of quorum systems are the introduction of improved hierarchical quorum systems [PM01], emulation of shared memory with the ability to change the used quorum system at run time [ES00], the introduction and theoretical analysis of byzantine quorum systems [MRW97], and a method to change the number of tolerated byzantine failures in a quorum system at run time [AMR+00].

The papers deal with different issues in the field of quorum systems. Combining the results of the papers, they form a powerful state of the art building block for quorum systems, which provides dynamically adapted tolerance to byzantine failures, a quorum-based shared memory as well as optimal load and availability of the quorum system.

References

[AMR+00] L. Alvisi, D. Malkhi, E. Pierce, M. K. Reiter, R. N. Wright: *Dynamic Byzantine Quorum Systems*, Proceedings of the International Conference on Dependable Systems and Networks, New York, 2000
Available at **`http://citeseer.nj.nec.com/alvisi00dynamic.html`**.

[MR98] D. Malkhi, M. Reiter: *Byzantine Quorum Systems*, Distributed Computing, 11(4): pp. 203–213, 1998
Available at **http://citeseer.nj.nec.com/malkhi97byzantine.html**.

[MRW97] D. Malkhi, M. Reiter, A. Wood: *The Load and Availability of Byzantine Quorum Systems*, Proceedings of the 16^{th} ACM Symposium on Principles of Distributed Computing (PODC), pp. 249–257, 1997
Available at **http://citeseer.nj.nec.com/malkhi97load.html**.

[ES00] B. Englert, A. A. Shvartsman: *Graceful Quorum Reconfiguration in a Robust Emulation of Shared Memory* Proceedings of. 20^{th} International Comference on Distributed Computing Systems (ICDCS-20), Taipei, Taiwan, 2000
Available at **http://www.math.ucla.edu/~englert/research/es.ps**.

[LS97] N. Lynch, A. Shvartsman: *Robust Emulation of Shared Memory Using Dynamic Quorum-acknowledged Broadcasts* Proceedings of the 27^{th} International Symposium on Fault Tolerant Computing Systems, 1997
Available at **http://theory.lcs.mit.edu/tds/papers/Lynch/FTCS97.html**.

[PM01] N. Preguiça, J. L. Martins: *Revisiting Hierarchical Quorum Systems*, Proceedings of the 21^{st} International Conference on Distributed Computing Systems (ICDCS-21), Phoenix, 2001
Available at **http://www-asc.di.fct.unl.pt/~nmp/papers/icdcs.pdf**.