b
UNIVERSITÄT
BERN

Generalized Quorums for Consensus

A Generalization of Byzantine quorum systems made practical

Bachelor Thesis

Angela Keller
from
Bern BE, Switzerland

Philosophisch-naturwissenschaftlichen Fakultät
der Universität Bern

21. January 2020

Prof. Dr. Christian Cachin
Orestis Alpos

Cryptology and Data Security Group
Institut für Informatik
University of Bern, Switzerland

Abstract

Quorum systems are overlapping sets of servers that ensure the consistency of systems. They are used to deal with failures in distributed systems. Even though the theory of Byzantine quorum systems exists for a long time, they are not really used in real life systems. In this work we show a way to make them practical. Our aim is to construct a system that tolerates more failures than $n > 3f$.

Contents

1 Introduction

In distributed trust, as the name suggests, we do not rely on a single node, instead we spread the trust amongst a group of nodes. This allows us to have failures and corruptions in a system but still get the right answer. Quorum systems are well known tools to implement the distributed trust in data repositories [5]. The traditional trust is the threshold trust, implemented by threshold quorum systems, which ensures the trust by numbers. In the threshold case we have n nodes in total and f faulty (Byzantine) nodes. This typically requires $n > 3f$. The threshold trust is symmetric, all nodes trust equally and all nodes are equally trusted.

In this paper we focus on Byzantine Quorum Systems. With Byzantine Quorum systems it is possible to build distributed systems that allow more than this previous $n > 3f$ failures. This trust is still symmetric, all nodes trust equally but not all nodes are equally trusted.

In this work we introduce the reader to quorum systems, including the Byzantine quorum systems. Additionally we want to create an algorithm for a generalized Byzantine broadcast protocol to show how Byzantine quorum systems are implemented in programs. The main purpose of this work is to reduce the amount of correct servers in quorum systems so that we can deal with more failures at the same time. For this we construct a system where we use attributes to differentiate servers in the system. With this differentiation and the rules we use we are able to construct smaller quorums without affecting the availability or the consistency of a system. In this way we try to create a theoretically correct system which allows more failures than the threshold case with $n > 3f$.

2 Preliminaries

2.1 Quorum Systems

Quorum systems according to Malkhi and Reiter [5] are well-known tools for ensuring the consistency and the availability of replicated data despite the benign failure of data repositories. Consistency in this context states that after a transaction in a distributed system every server should have the same information. Availability states that the servers should respond in an acceptable amount of time. There are different ways to construct such quorums. In this work the focus lies on Byzantine quorum systems which are explained in section 2.2. To work with quorum systems we assume that we have a universe of servers U. A quorum system then is a set of subsets of U which includes the individual quorums. According to the definition of quorum systems each of these quorums intersect.

Definition 1. Let the universe U be a set of servers. A *quorum system* $\mathcal{Q} \subseteq 2^U$ is a non-empty set of subsets of U, every pair of which intersects. Each $Q \in \mathcal{Q}$ is called a *quorum*.

2.1.1 Majority Quorum System

In the majority quorum system [1] the majority of the servers are assumed to be correct. That means that it tolerates $f < \frac{n}{2}$ faulty servers. A quorum in this case contains $\frac{n+1}{2}$ servers. It is possible to construct weighted majority by assigning multiple votes to some servers in the system.

Example 1. Imagine a system with 20 servers, so $n = 20$. The maximum number of faulty servers can be calculated as follows

$$f < \frac{20}{2}$$
$$f < 10$$
$$f = 9$$

2.1.2 Grid Quorum System

In a grid quorum system [1] we have $k \cdot k = n$ servers that are arranged in a square. A quorum is a full row and one element out of each row below that full row.

Example 2. Imagine a system with $k = 5$, so $n = 5 \cdot 5 = 25$ servers. In this example we have all the servers in the second row and one server in each row below in the quorum, as it is shaded in figure 2.1. This example works because it doesn't matter which row we take as a starting point, when we take a quorum with this specifications it will always intersect in at least one server from another quorum.

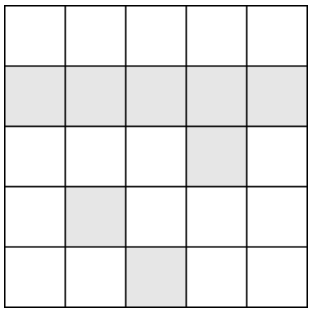

Figure 2.1. A quorum system on $n = 25$ servers.

2.2 Byzantine Quorum System

To introduce the Byzantine Quorum Systems we will use the construction of the Dissemination quorum systems from Malkhi and Reiter [5]. We assume that we have a universe U of servers, where $|U| = n$, and an arbitrary number of clients that are distinct from the servers. A *quorum system* $\mathscr{Q} \subseteq 2^U$ is a non-empty set of subsets of U, every pair of which intersect. Each $Q \in \mathscr{Q}$ is called a *quorum*.

The servers and clients that obey their specifications are *correct*. The *faulty* servers that exist may deviate from it's specification arbitrarily. A *fail-prone system* $\mathscr{B} \subseteq 2^U$ is a

non-empty set of subsets of U, none of which is contained in another, such that some $B \in \mathscr{B}$ contains all the *faulty* servers [5].

Definition 2. *A quorum system $\mathscr{Q}$ is a* Byzantine quorum system *for a fail-prone system $\mathscr{B}$ if the following properties are satisfied [5].*

Consistency: $\forall\, Q_1, Q_2 \in \mathscr{Q}\ \forall B \in \mathscr{B} : Q_1 \cap Q_2 \nsubseteq B$
Availability: $\forall B \in \mathscr{B}\ \exists\, Q \in \mathscr{Q} : B \cap Q = \emptyset$

Theorem 1. *Let $\mathscr{B}$ be a fail-prone system for a universe U. Then there exists a Byzantine quorum system for $\mathscr{B}$ iff (if and only if) $\mathscr{Q} = \{U \setminus B : B \in \mathscr{B}$ is a Byzantine quorum system for $\mathscr{B}$*[5].

Corollary 2. *Let $\mathscr{B}$ be a fail-prone system for a universe U. Then there exists a Byzantine quorum system for $\mathscr{B}$ iff for all $B_1, B_2, B_3 \in \mathscr{B}, U \nsubseteq B_1 \cap B_2 \cap B_3$. In particular (that would be the threshold Byzantine quorum system) suppose that $\mathscr{B} = \{B \subseteq U : |B| = f\}$. Then there exists a Byzantine quorum system for $\mathscr{B}$ iff $n > 3f$* [5].

Example 3. Garcia-Perez and Gotsman [4] provided in their work from 2018 an example of such a Byzantine quorum system with $n = 4$. Consider the universe $U = \{1, 2, 3, 4\}$, the quorum system $\mathscr{Q} = \{\{1, 2\}, \{1, 2, 3\}, \{1, 3, 4\}, \{1, 2, 3, 4\}\}$ and a fail-prone system $\mathscr{B} = \{\{2\}, \{3, 4\}\}$.

Then $(\mathscr{Q}, \mathscr{B})$ is a Byzantine quorum system. **Consistency** holds because all quorums in $\mathscr{Q}$ intersect at 1, which does not belong to any element of $\mathscr{B}$. **Availability** holds because both $\{1, 2\}$ and $\{1, 3, 4\}$ are the minimal quorums. This can be proved with definition 2 in this chapter.

2.2.1 The threshold Byzantine quorum system

The threshold case for the Byzantine quorum system is that we have a system that tolerates $n > 3f$ faulty servers. In the threshold case we can simply calculate how many faulty servers for each system size can be tolerated. Also we can derive how big the quorum size has to be and how many servers in a quorum have to be correct ones. We have 3 properties which have to hold.

Definition 3. Two Byzantine quorums intersect in at least one correct server.

Definition 4. There is at least one quorum with only correct servers.

Definition 5. The majority of a Byzantine quorum has to be correct servers.

We have a system with n servers and f faulty servers. The number of correct nodes is therefore $n - f$. A Byzantine quorum is a set of more than $\frac{n+f}{2}$ servers.

Example 4. Imagine a system with 7 servers. So we know that $n = 7$, $n > 3f$ and the Byzantine quorum has to be greater than $\frac{n+f}{2}$.

$$n > 3f \Leftrightarrow f < \frac{n}{3}$$

For $n = 7$, we get

$$f < \frac{7}{3}$$
$$f = 2$$

This calculation gives us the number of faulty servers in a system of $n = 7$. Let us now calculate how many servers are in a quorum. We need at least

$$\frac{n+f}{2}$$

servers. So for $n = 7, f = 2$ we get

$$> \frac{7+2}{2}$$
$$> \frac{9}{2}$$
$$= 5$$

The quorum size in this example therefore is 5. Now we want to calculate how many servers in a Byzantine quorum are correct ones. For this we can take the number of servers that are in a Byzantine quorum and subtract the faulty ones.

$$> \frac{n+f}{2} - f$$
$$= \frac{n+f-2f}{2} = \frac{n-f}{2}$$

$$> \frac{n-f}{2}$$

Now we know that the number of correct servers c in a Byzantine quorum has to be greater than $\frac{n-f}{2}$. In our example this would mean:

$$c > \frac{n-f}{2}$$

For $n = 7, f = 2$:

$$c > \frac{7-2}{2}$$

$$c > \frac{5}{2}$$

which gives

$$c = 3$$

The number of correct servers in a quorum from this example therefore is 3. In figure 2.2 we now see an example of two quorums in a system with $n = 7$. The faulty servers are marked with a cross. We can see that the quorum size is 5 as calculated, the number of correct servers in a quorum is 3. When we intersect these two quorums we also see that there is one correct server in the intersection. We also can take the 5 upper servers from the figure and create a quorum with only correct servers. This proves all the definitions above and therefore this example is a working Byzantine quorum system.

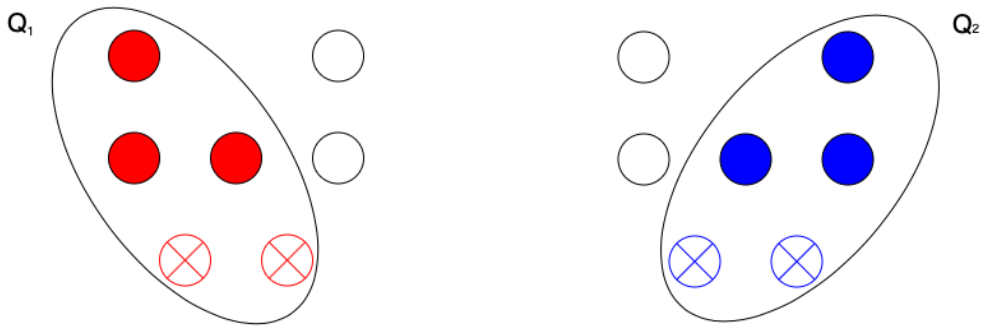

Figure 2.2. Quorum Q_1 and Q_2 of a system with 7 servers.

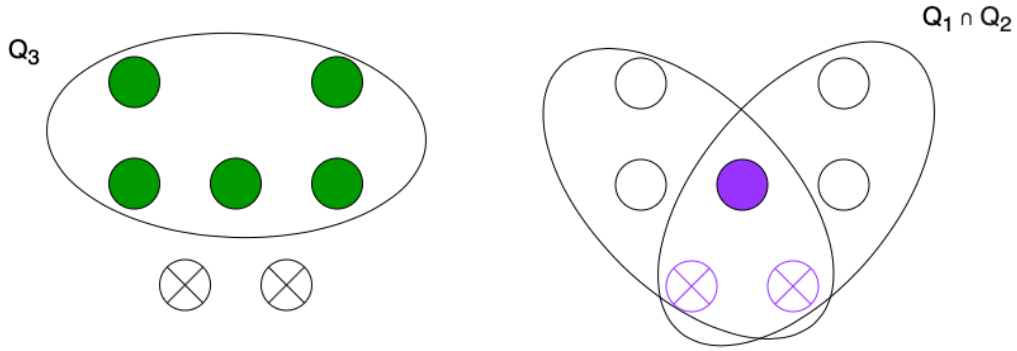

Figure 2.3. Quorum Q_3 and $Q_1 \cap Q_2$ of a system with 7 servers.

2.3 Differentiating Servers by Attributes

In his work that was published in the 2001 International Conference on Dependable Systems and Networks [2], Cachin constructed an example of a Byzantine quorum system where he used the location and the operating system of the servers as attributes. To understand the following example we need to introduce some notation. Let $class$ denote the name of an attribute and also the set of attribute values. For $c \in class$, let $\chi_c : 2^P \to \{0,1\}$ be the characteristic function of the attribute on a set of parties, i.e., $\chi_c(S) = 1$ if and only if $class(i) = c$ for some $i \in S$ [2]. Θ describes how many out of how many have to be true. Let $class_1 = \{a, b, c, d\}$ describe the locations and let $class_2 = \{\alpha, \beta, \gamma, \delta\}$ describe the operating systems. The distributed system in this example tolerates the simultaneous failure of all the servers in one location and all the servers of one operating system. To check if the properties hold we also need to check the Q^3 condition, which states that three sets in $\mathscr{B}$ will not cover the universe of servers U. The arbitrary structure in this case is characterized as follows:

$$g(S) = \overline{\Theta_2^4(x_a, x_b, x_c, x_d)} \vee \overline{\Theta_2^4(y_\alpha, y_\beta, y_\gamma, y_\delta)}$$

where

$$x_v = \Theta_2^4(\chi_v(S) \wedge \chi_\alpha(S), \chi_v(S) \wedge \chi_\beta(S), \chi_v(S) \wedge \chi_\gamma(S), \chi_v(S) \wedge \chi_\delta(S))$$

for $v \in class_1$, and

$$y_\nu = \Theta_2^4(\chi_a(S) \wedge \chi_\nu(S), \chi_b(S) \wedge \chi_\nu(S), \chi_c(S) \wedge \chi_\nu(S), \chi_d(S) \wedge \chi_\nu(S))$$

for $\nu \in class_2$.

This equation explained in words tells us that there are not two out of 4 locations, where 2 or more operating systems fail at the same time. So there can just be one location where 2 or more servers fail at the same time. The first Θ of the first equation states that there cannot be 2 out of 4 operating systems where 2 or more locations fail. So there can just be one operating system where 2 or more servers at the same time fail. The quorum would be the negation of the arbitrary structure. This would be $\overline{g(S)} = \Theta_2^4(x_a, x_b, x_c, x_d) \vee \Theta_2^4(y_\alpha, y_\beta, y_\gamma, y_\delta)$. All servers that are not in this previous described arbitrary structure form a quorum.

Example 5. We arrange the servers in a grid where elements $class_1$ states the locations and elements of $class_2$ states the operating system. In the upper left corner of figure 2.2. we assume that all the servers in location b (Tokyo) fail, and every operating system beta (Windows NT). So in this case the fail-prone set is:

$$\{a\beta, b\alpha, b\beta, b\gamma, b\delta, c\beta, d\beta\}$$

The quorum in this case is:

$$\{a\alpha, a\gamma, a\delta, c\alpha, c\gamma, c\delta, d\alpha, d\gamma, d\delta\}$$

In the upper right corner of figure 2.2. we crossed every location c (Zurich) out, and every operating system γ (Linux). So in this case the fail-prone set is:

$$\{a\gamma, b\gamma, c\alpha, c\beta, c\gamma, c\delta, d\gamma\}$$

The quorum in this case is:

$$\{a\alpha, a\beta, a\delta, b\alpha, b\beta, b\delta, d\alpha, d\beta, d\delta\}$$

In the lower left corner of figure 2.2. we crossed every location d (Haifa) out, and every operating system α (AIX). So in this case the fail-prone set is:

$$\{a\alpha, b\alpha, c\alpha, d\alpha, d\beta, d\gamma, d\delta\}$$

The quorum in this case is:

$$\{a\beta, a\gamma, a\delta, b\beta, b\gamma, b\delta, c\beta, c\gamma, c\delta\}$$

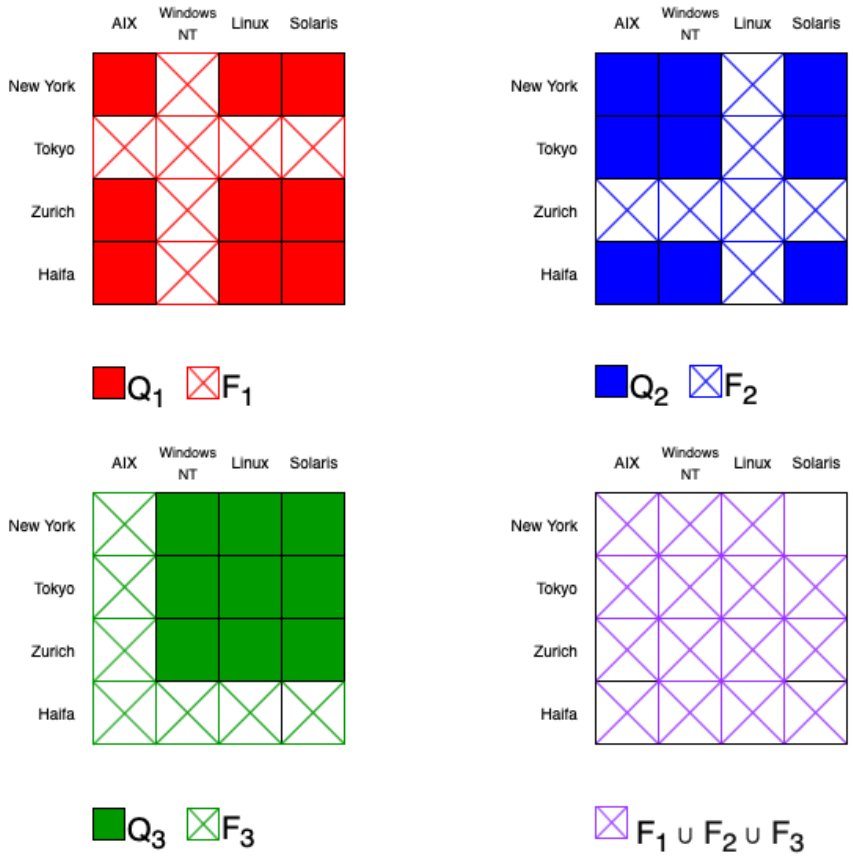

Figure 2.4. Three fail-prone sets and three quorums on $n = 16$ servers and the union of three fail-prone sets to check the Q^3 condition.

The following figure shows these three different quorums and their fail-prone sets like they are described above. To check the Q^3 condition we added the union of these three fail-prone sets. We can see that there is still a server left that is not included in this union, therefore the Q^3 holds. In chapter 4 of this work, we will also prove the availability and the consistency for similar examples.

3

A generalized Byzantine broadcast protocol

3.1 Byzantine Consistent Broadcast

With the system model from Dahlia Malkhi and Michael Reiter [5] here we generalize an algorithm from the Reliable and Secure Distributed Programming book [3] that was constructed for the threshold Byzantine quorum system $n > 3f$. This gives us an idea how an algorithm constructed with the Byzantine Quorum System works. The algorithm we present is the adapted Signed Echo Broadcast algorithm 3.17 [3] which uses an authenticated perfect links abstraction and a cryptographic digital signature scheme.

In this algorithm the sender sends a message m to every server in the system and then waits for an answer from the receivers of the message. The receivers have to sign the message and return it to the sender. The sender expects a Byzantine quorum of servers to witness the message.

On the next page there is the definition we use for this algorithm. The events we use are the request for the broadcast and the indication for the deliver. Next there are four properties that have to hold for the Byzantine consistent broadcast. The first property is the *Validity*, which states that when a correct process p broadcasts a message m then every correct process eventually delivers m. Second we have the *No duplication* property that states that every correct process delivers not more than one message. Third the

Integrity which states that if some correct process delivers a message m with sender p and process p is correct, then m was previously broadcast by p. And last we have the Consistency property which states that if some correct process delivers a message m and another correct process delivers a message m', then $m = m'$ [3].

Module 1 Interface and properties of Byzantine consistent broadcast [3]

Module:

Name: ByzantineConsistentBroadcast, **instance** *bcb*, with sender s.

Events:

Request: $\langle$ *bcb, Broadcast* | m $\rangle$: Broadcasts a message m to all processes. Executed only by process s

Indication: $\langle$ *bcb, Deliver* | p, m $\rangle$: Delivers a message m broadcast by process p

Properties:

BCB1: *Validity:* If a correct process p broadcasts a message m, then every correct process eventually delivers m.

BCB2: *No duplication:* Every correct process delivers at most one message.

BCB3: *Integrity:* If some correct process delivers a message m with sender p and process p is correct, then m was previously broadcast by p.

BCB4: *Consistency:* If some correct process delivers a message m and another correct process delivers a message m', then $m = m'$.

Algorithm 1 Generalized Signed Echo Broadcast

Implements:
ByzantineConsistentBroadcast, **instance** *bcb*, with sender s.

Users:
AuthPerfectPointToPointLinks, **instance** al.

upon event $\langle$ *bcb, Init* $\rangle$ **do**
sentecho := FALSE;
sentfinal := FALSE;
delivered := FALSE;
echos := $\left[\perp\right]^N$; $\Sigma := \left[\perp\right]^N$

upon event $\langle$ *bcb, Broadcast* | m $\rangle$ **do** // only process s
forall $q \in \Pi$ **do**
trigger send message $\langle$ $al, Send$ | q, $\left[\text{SEND, } m\right]$ $\rangle$;

upon receiving message $\langle$ $al, Deliver$ | p, $\left[\text{SEND, } m\right]$ $\rangle$ **such that** $p = s$ **and** *sentecho* = FALSE **do**
sentecho := TRUE;
σ := *sign*(*self*, *bcb*||*self*||ECHO||*m*)
trigger send echo $\langle$ $al, Send$ | s, $\left[\text{ECHO, } m, \sigma\right]$ $\rangle$;

upon receiving echo $\langle$ $al, Send$ | p, $\left[\text{ECHO, } m, \sigma\right]$ $\rangle$ **do** // only process s
if *echos*[p] = $\perp \wedge$ *verifysig*(p, *bcb*||p||ECHO||m,σ) **then**
echos[p] := m; Σ [p] := σ;

upon exists $m \neq \perp$ **such that** $\{p \mid echos[p] = m\} \in \mathcal{Q}$ **and** *sentfinal* = FALSE **do**
sentfinal := TRUE;
forall $q \in \Pi$ **do**
trigger $\langle$ $al, Send$ | q, $\left[\text{FINAL, } m, \Sigma\right]\rangle$;

upon event $\langle$ $al, Deliver$ | p, $\left[\text{FINAL, } m, \Sigma\right]$ $\rangle$ **do**
if $\{p \mid \Sigma[p] \neq \perp \wedge$ *verifysig*(p, *bcb*||p||ECHO||m, $\Sigma[p]$)$\} \in \mathcal{Q}$
and *delivered* = FALSE **do**
delivered := TRUE;
trigger $\langle$ *bcb, Deliver* | $s, m\rangle$;

4 Generalized Quorums made practical

4.1 Differentiating servers by location and operating system

To construct generalized quorum systems that could actually be used in real word environments we build on the work of Cachin [2]. There he constructs an example with a system of 16 servers with two different attributes. The attributes he used are the location and the operating system. This example can also be found in the preliminaries of this thesis in chapter 2. The idea of this system is, that when we have servers at each location with an amount of different operating systems, and all servers of a specific operating system and all servers in a specific location fail or are corrupted then the other operating systems and locations should most likely not fail at the same time.

4.1.1 Seven locations and four operating systems

The first step we have done to adapt the theory to a new example is that we created a system with 28 servers. We also use the location and the operating system as attributes. In this case we added three new locations to the system. So we still have four operating systems, but we have seven different locations.

The first question we want to answer about this system is how many simultaneously faulty servers it tolerates, which means how big a fail-prone set can be. If we look at it from the threshold perspective we know that $f < \frac{n}{3}$ [5]. With 28 servers this means that

we could have no more than nine servers in a fail-prone set.

But with the generalization we can have more than these nine servers which fail at the same time. The system is constructed in the way, that at the same time each server with one operating system can fail and each server of two different locations. This system therefore allows 13 servers which fail at the same time.

Example 6. The attributes are arranged in two classes. We have $class_1$ with $C_1 = \{OS_1, OS_2, OS_3, OS_4\}$ with the operating systems Windows Server, Mac OSX Server, Red Hat Enterprise Linux, and SUSE Linux Enterprise Server. Then we have $class_2$ with $C_2 = \{L_1, L_2, L_3, L_4, L_5, L_6, L_7\}$, which are the locations Virginia (USA), Tokyo (Japan), Zurich (Switzerland), Haifa (Israel), Hong Kong (China), Dublin (Ireland), and Sao Paolo (Brazil). Arranged in a grid this looks as follows:

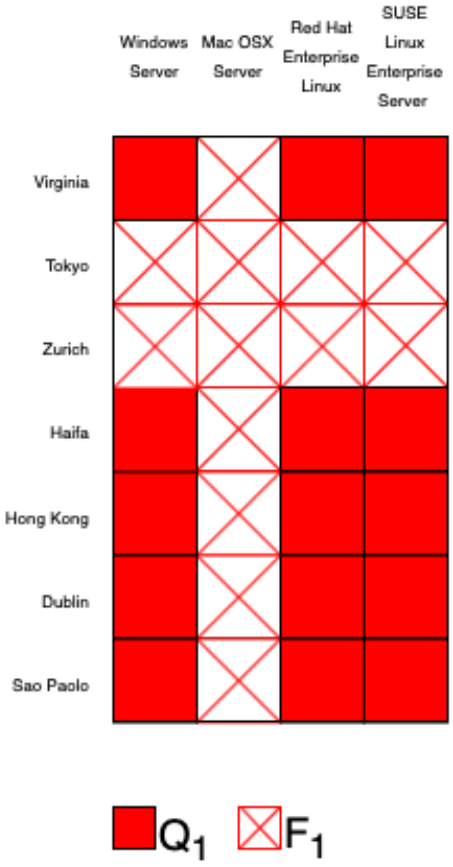

Figure 4.1. Quorum Q_1 and the fail-prone set F_1 of a system with 28 servers in a grid

Imagine now we have two other fail-prone sets with two columns and one row crossed out. When we would add them in the grid, there would still be one server left that isn't in the union of these three fail prone systems. So we checked the Q^3 condition [2]. We can also check the consistency by arranging the servers in a tree structure.

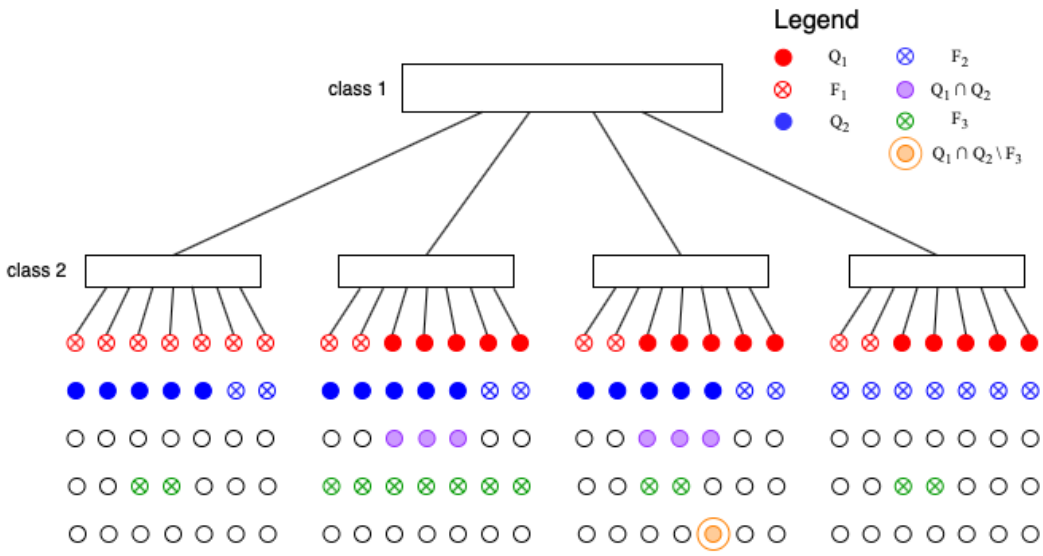

Figure 4.2. Quorums Q_1, Q_2 and the fail-prone sets F_1, F_2, F_3, intersection $Q_1 \cap Q_2$ of a system with 28 servers in a tree.

We now see that $Q_1 \cap Q_2 \setminus F_3 \neq \emptyset$. This proves that the consistency is satisfied. The proof for the availability is even simpler, because for each fail-prone system we have its associated quorum and they do not intersect with each other. As an example for the availability we can check Q_1 and F_1 together.

Now we want to give a mathematical definition of the fail-prone system. The negation of this definition then will be the quorum system.

Definition 6. The tuple of one attribute from $class_1$ and one attribute from $class_2$ describe a server at a specific location with a specific operating system.

Let $class_1$ and $class_2$ be:

$$C_1 = \{OS_1, OS_2, OS_3, OS_4\}$$

$$C_2 = \{L_1, L_2, L_3, L_4, L_5, L_6, L_7\}$$

The universe of servers U is the Cartesian product of all attributes from $class_1$ and $class_2$:

$$U = C_1 \times C_2$$

The fail-prone sets can be described as follows:

$$F_{i,j_{1,2}} = \{OS_i\} \times C_2 \quad \cup \quad C_1 \times \{L_{j_1}\} \quad \cup \quad C_1 \times \{L_{j_2}\}$$

with $OS_i \in C_1$ and $L_{j_{1,2}} \in C_2$ where $j_1 \neq j_2$

When we now take this definition and the example 6 from this chapter the fail-prone set F_1 would be described as follows:

$$F_1 = \{OS_2\} \times C_2 \quad \cup \quad C_1 \times \{L_2\} \quad \cup \quad C_1 \times \{L_3\}$$

4.1.2 Seven locations and seven operating systems

In the second step we added 3 more operating systems so that we have a system with 49 servers. In this example we have 7 locations and 7 operating systems. Also here the first question we want to answer is how many servers can fail at the same time. In the threshold case we know the equation is $f < \frac{n}{3}$ [5]. In the example with 49 servers this would mean no more than 16 servers can fail at the same time.

But how does it look with generalization? We constructed the system in the way that at the same time two locations and two operating systems can fail. This system then tolerates 28 simultaneous failures.

Example 7. The attributes are arranged in two classes. $class_1$ with the operating systems Windows Server, Mac OSX Server, Red Hat Enterprise Linux, SUSE Linux Enterprise Server, Free BSD, Solaris and Sun Cobalt. Then we have $class_2$ with the locations Virginia (USA), Tokyo (Japan), Zurich (Switzerland), Haifa (Israel), Hong Kong (China), Dublin (Ireland), and Sao Paolo (Brazil). Arranged in a grid this looks as follows:

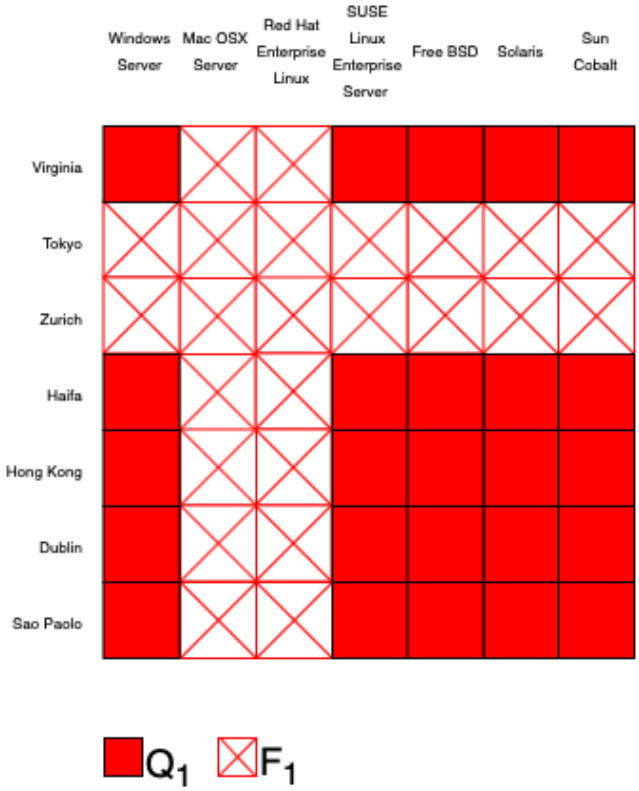

Figure 4.3. Quorum Q_1 and fail-prone set F_1 of a system with 49 servers in a grid.

Imagine now we have 2 other fail-prone systems with two columns and two lines crossed out. When we add them in the grid, there would still be one server left that isn't in the union of these three fail prone systems. So we checked the Q^3 condition [2]. We can also check the consistency by arranging the servers in a tree structure.

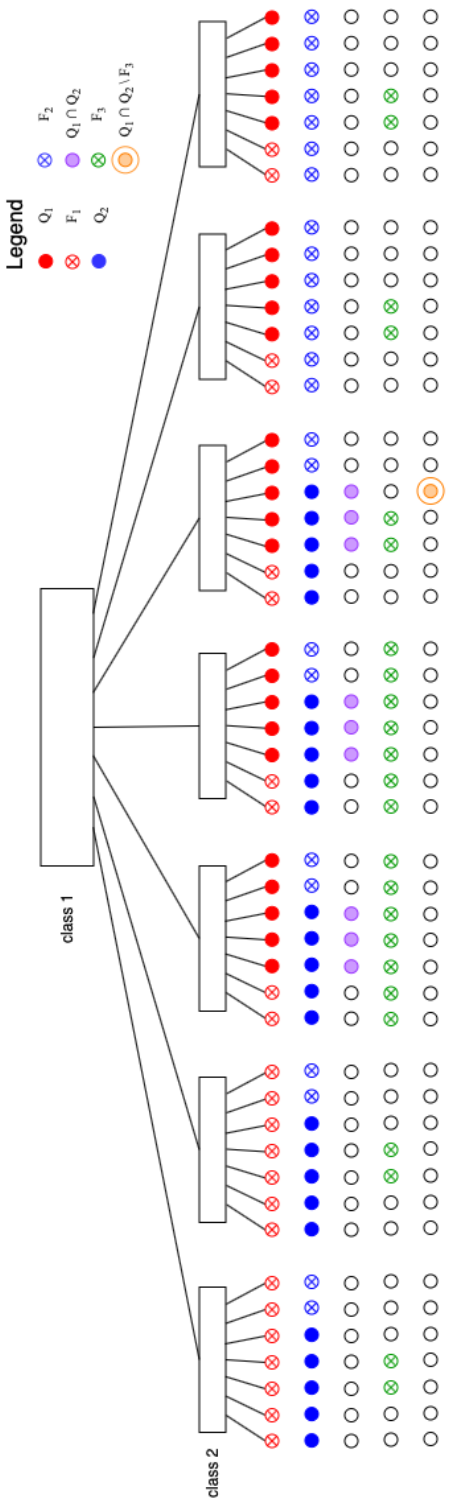

Figure 4.4. Quorums Q_1, Q_2 and fail-prone sets F_1, F_2, F_3, intersection $Q_1 \cap Q_2$ of a system with 49 servers in a tree.

We now see that $Q_1 \cap Q_2 \setminus F_3 \neq \emptyset$. This proves that consistency is satisfied. The proof for availability is even simpler, because for each fail-prone system we have its associated quorum and they do not intersect with each other. As an example for the availability we can just check Q_1 and F_1

Definition 7. The tuple of one attribute from $class_1$ and one attribute from $class_2$ describe a server at a specific location with a specific operating system.

Let $class_1$ and $class_2$ be:

$$C_1 = \{OS_1, OS_2, OS_3, OS_4, OS_5, OS_6, OS_7\},$$
$$C_2 = \{L_1, L_2, L_3, L_4, L_5, L_6, L_7\}$$

The universe of servers U is the Cartesian product of all attributes from $class_1$ and $class_2$:

$$U = C_1 \times C_2$$

The fail-prone sets can be described as follows:

$$F_{i_{1,2},j_{1,2}} = \{OS_{i_1}\} \times C_2 \quad \cup \quad \{OS_{i_2}\} \times C_2 \quad \cup \quad C_1 \times \{L_{j_1}\} \quad \cup \quad C_1 \times \{L_{j_2}\}$$

with $\quad OS_{i_{1,2}} \in C_1, \quad i_1 \neq i_2 \quad$ and $\quad L_{j_{1,2}} \in C_2 \quad , \quad j_1 \neq j_2$

When we now take this definition and the example 7 from this chapter the fail-prone set F_1 would be described as follows:

$$F_1 = \{OS_2\} \times C_2 \quad \cup \quad \{OS_3\} \times C_2 \quad \cup \quad C_1 \times \{L_2\} \quad \cup \quad C_1 \times \{L_3\}$$

4.1.3 Scalability of a system with two attribute classes

We now show two different examples with an distinct number of attributes per class. Can the system be constructed by rules which work for every amount of different attributes in these classes? For this we searched a rule how the system can be built that it also works when we scale it. We know that the Q^3 condition [2] can only be satisfied when we have more than 3 times the amount of rows that are in the fail-prone system and the same for the columns. Because when we add two other fail-prone sets we must still have servers that are not in this fail-prone system. The number of the rows in this case would be the amount of attributes in $class_1$ and the amount of columns would be the amount of attributes in $class_2$. When we would exactly have 3 times the amount of rows that are in a fail-prone set the Q^3 condition [2] would not hold, because the union of 3 fail-prone sets would cover all the servers in the system.

We want to formulate this rule mathematically, so that we can just calculate the number of possible server failures for each system size.

Theorem 3. *Imagine a system with n attributes in $class_1$ and m attributes in $class_2$, where n is the amount of rows and m is the amount of columns.*

The number of possible faulty rows x' can be calculated as follows:

$$n = 3x + 1$$

where $x' = floor(x)$

The number of possible faulty columns y' can be calculated as follows:

$$m = 3y + 1$$

where $y' = floor(y)$

The size of a fail-prone set f can be calculated as follows:

$$y' \cdot n + x' \cdot m - x' \cdot y'$$

We produced a table and use this calculations to show a few systems with different numbers of servers and their quorum and fail-prone set sizes. With this calculation we could take any amount of attributes in this two classes and we would find a working Byzantine quorum system.

class$_1$	class$_2$	servers s	faulty rows	faulty cols	failures f	quorum q
n	m	$n * m$	x'	y'	f	$s - f$
4	4	16	1	1	7	9
4	7	28	1	2	13	15
7	7	49	2	2	24	25
10	10	100	3	3	51	49

Table 4.1. Systems with two attribute classes and a various amount of attributes per class.

4.2 Differentiating servers by cloud provider, location and operating system

In the next step we added one attribute class more to the system for the 3rd dimension, the cloud provider. This system we will not present as a grid because with the three dimensions we could not really analyse it in the grid structure. In this example we have four cloud providers, four operating systems and four locations. This gives us a total of 64 servers in this system, where one server is a tuple with three attributes, one attribute from each class. Also the first thing we analyse here is the threshold case where we have $f < \frac{n}{3}$ [5] possible failures. In the threshold case we can therefore have 21 failures at the same time. But when we build the system with generalization then we reach the possible amount of 37 failures at the same time.

Example 8. The attributes are arranged in three classes. $class_1$ with the Cloud Providers AWS (Amazon), Azure (Microsoft), Google Cloud (Google), IBM Cloud (IBM). $class_2$ with the operating systems Windows Server, Mac OSX Server. Red Hat Enterprise Linux and SUSE Linux Enterprise Server. $class_3$ with the locations Virginia (USA), Tokyo (Japan), Zurich (Switzerland) and Haifa (Israel).

In the tree structure of this system we have one class more and therefore we have one dimension more in the tree. In this example we have in F_1 the Cloud Provider AWS (Amazon), the operating system Windows Server and the location Virginia (USA) which all fail at the same time. To construct F_1 we always take the nodes from the left branches in the tree and added these servers in the fail-prone set. For F_2 we took all the nodes from the right branches of every class and added them to the fail-prone set. The quorums Q_1 and Q_2 then are $U - F_1$ and $U - F_2$. In F_3 all servers from the second branches of the tree are included.

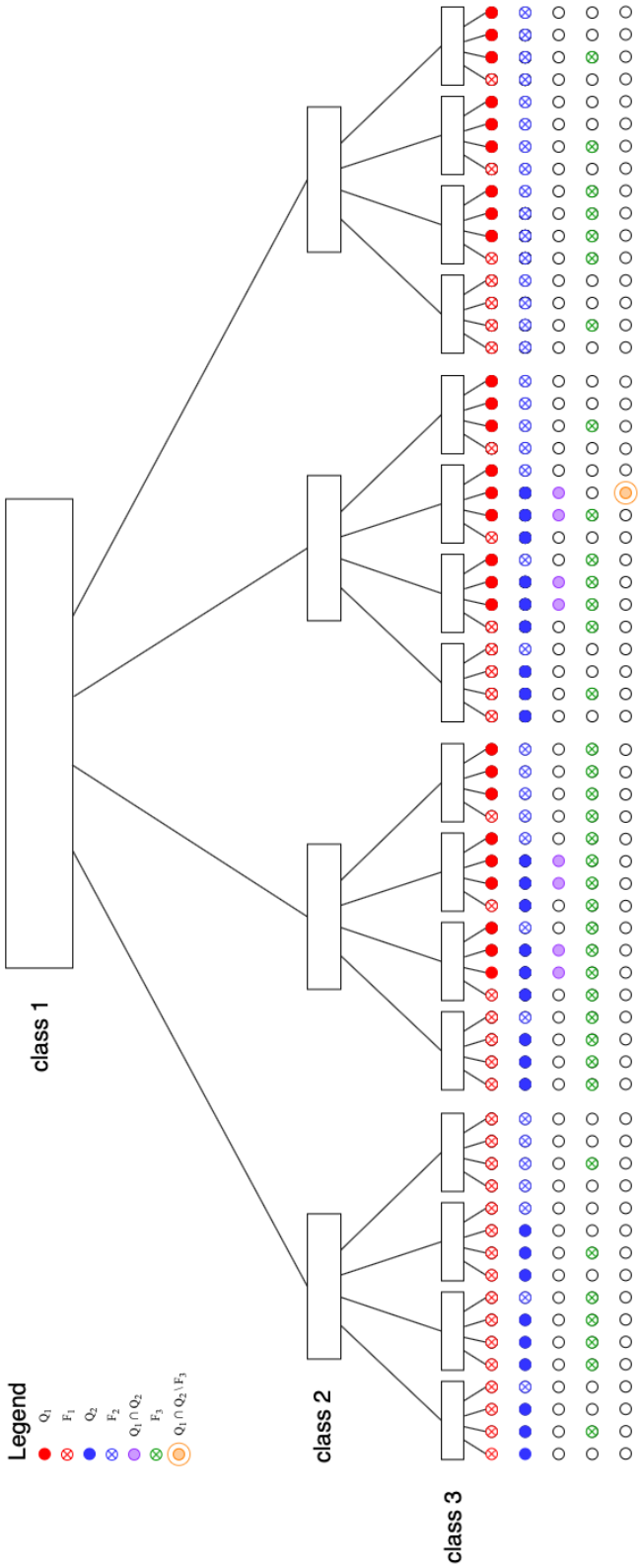

Figure 4.5. Quorum Q_1, Q_2 and fail-prone sets F_1, F_2, F_3, intersection $Q_1 \cap Q_2$ of a system with 64 servers in a tree.

Definition 8. The tuple of one attribute from $class_1$, one attribute from $class_2$ and one attribute from $class_3$ describes a server with a specific cloud provider, a specific operating system at a specific location.

Let $class_1$, $class_2$ and $class_3$ be:

$$C_1 = \{CP_1, CP_2, CP_3, CP_4\}$$
$$C_2 = \{OS_1, OS_2, OS_3, OS_4\}$$
$$C_3 = \{L_1, L_2, L_3, L_4\}$$

The universe of servers U is the Cartesian product of all attributes from $class_1$, $class_2$ and $class_3$:

$$U = C_1 \times C_2 \times C_3$$

The fail-prone sets can be described as follows:

$$F_{i,j,k} = \{CP_i\} \times C_2 \times C_3 \quad \cup \quad C_1 \times \{OS_j\} \times C_3 \quad \cup \quad C_1 \times C_2 \times \{L_k\}$$

with $CP_i \in C_1$, and $OS_j \in C_2$, and $L_k \in C_3$

When we now take this definition and the example 8 from this chapter the fail-prone set F_1 would be described as follows:

$$F_1 = \{CP_1\} \times C_2 \times C_3 \quad \cup \quad C_1 \times \{OS_1\} \times C_3 \quad \cup \quad C_1 \times C_2 \times \{L_1\}$$

4.3 General description of the arbitrary structure of the generalized quorum system

In this section we show the general description to construct such a Byzantine Quorum System of any size. This means that we can have any number of classes and any number of attributes inside of these classes.

The idea will always be the same for every system size. We can arrange our servers with the previous used tree structure. As we get more classes, the amount depth of the tree increases. As we get more attributes per class, the tree grows wider. To construct our arbitrary structure we always follow the same rules, which change a little bit regarding the number of attributes per class.

The smallest system size where our construction works is with four attributes per class and two attribute classes. The idea here is that we decide for each class which branch we add to our arbitrary structure. Per class the branch must be always the same, because it has to be the same attribute (For example all the servers with the same operating system to one fail-prone set). We number our branches from one to four. For the first and second class we then have four possibilities which branch we add to the arbitrary structure.

Example 9. For two classes with four attributes per class we can construct 16 different fail-prone sets. We can choose 4 different branches for the first class and then again four different branches for the second class. This is exactly the number of permutations $n * m$ where n is the number of attributes in $class_1$ and m is the number of elements in $class_2$.

When the number of attributes per class gets higher, we have to check with the formula from theorem 3, if we can add more than one branch to the arbitrary structure. As we know, with seven attributes per class we have two branches to add because it is possible that for example two operating systems fail at the same time and not just one like in the case with 4 attributes per class. With ten attributes per class we can add three branches from the tree, and so on.

Example 10. For three classes with four attributes per class we can construct 64 different fail-prone sets. We can choose 4 different branches for the first class and then again four different branches for the second and third class. This is exactly the number of combinations $n * m * l$ where n is the number of attributes in $class_1$, m is the number of elements in $class_2$ and l is the number of attributes in $class_3$.

The system in general would be defined as follows:

First we have the construction of the classes, from $class_1$ to $class_N$:

$$
\begin{aligned}
C_1 &= \{a_1, a_2, a_3, ..., a_n\}, \quad a_i \in C_1 \\
C_2 &= \{b_1, b_2, b_3, ..., b_m\}, \quad b_j \in C_2 \\
C_3 &= \{c_1, c_2, c_3, ..., c_l\}, \quad c_k \in C_3 \\
&... \\
C_N &= \{N_1, N_2, N_3, ..., N_z\}, \quad N_w \in C_N
\end{aligned}
$$

The arbitrary structure can then be described as follows:

$$\begin{aligned} F_{i,j,k,\ldots,N} = &\{a_i\} \times C_2 \times C_3 \times \ldots \times C_N \quad \cup \quad C_1 \times \{b_j\} \times C_3 \times \ldots \times C_N \\ &\cup \quad C_1 \times C_2 \times \{c_k\} \times \ldots \times C_N \quad \cup \quad \ldots \quad \cup \quad C_1 \times C_2 \times C_3 \times \ldots \times \{N_w\} \end{aligned} \tag{4.1}$$

The construction follows from the idea described above. For each class we decide which branch we add to a fail-prone set. The branches in our case would describe either a cloud provider, an operating system or a location. But it does not matter with which kind of attributes the system is built. The construction stays the same.

5 Conclusion

Quorum systems are a well known way to work with replicated data and their consistency and availability in distributed systems. The goal we had for this work was to generalize quorum systems, while protecting their consistency and availability, so that we can step away from the threshold quorum that works by numbers.

We created a system that allows us to deal with more than the threshold case with $n > 3f$. For this we build a system where one server is a tuple of different attributes. This allowed us to construct rules, on which our Byzantine quorum system is build on. With these rules we are no longer in a system model where all servers are equally trusted.

We defined combinations of servers that can be trusted as a Byzantine quorum. These Byzantine quorums have to contain fewer correct servers than the threshold quorum would contain and they still meet all the conditions that are prescribed in the theory of Byzantine quorum systems.

In a next step a prototype with those specifications can be built and the model can be tested practically. With such a prototype the advantages such a system would bring us can be measured.

The research in the field of quorum systems then goes further to the asymmetric quorum systems, where the servers are not equally trusted, neither they trust equally. Such research is ongoing at the moment at the University of Bern.

Bibliography

[1] Christian Cachin. Security and fault-tolerance in distributed systems. Course Spring 2009, ETHZ, `www.zurich.ibm.com/~cca`.

[2] Christian Cachin. Distributing trust on the internet. In *2001 International Conference on Dependable Systems and Networks (DSN 2001) (formerly: FTCS), 1-4 July 2001, Göteborg, Sweden, Proceedings*, pages 183–192, 2001.

[3] Christian Cachin, Rachid Guerraoui, and Luís E. T. Rodrigues. *Introduction to Reliable and Secure Distributed Programming (2. ed.)*. Springer, 2011.

[4] Álvaro García-Pérez and Alexey Gotsman. Federated byzantine quorum systems. In *22nd International Conference on Principles of Distributed Systems, OPODIS 2018, December 17-19, 2018, Hong Kong, China*, pages 17:1–17:16, 2018.

[5] Dahlia Malkhi and Michael K. Reiter. Byzantine quorum systems. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on the Theory of Computing, El Paso, Texas, USA, May 4-6, 1997*, pages 569–578, 1997.