Byzantine Fault Tolerance
From Static Selection to Dynamic Switching

by
Ali Shoker

November 29, 2012

Submitted to the departement of Computer Science and Telecommunication in partial fulfillment of the requirements for the degree of Doctor of Philosophy with the European label at the University of Toulouse III and the Lebanese University.

This thesis was supervised by:
Jean-Paul Bahsoun, University of Toulouse, France.
Kablan Barbar, Lebanese University, Lebanon.

co-advised by:
Rachid Guerraoui, EPFL, Switzerland.

and examined by:
Miguel Correia, University of Lisbon, Portugal.
Hermann De Meer, University of Passau, Germany.

Byzantine Fault Tolerance
From Static Selection to Dynamic Switching

by
Ali Shoker

Abstract

Byzantine Fault Tolerance (BFT) is becoming crucial with the revolution of online applications and due to the increasing number of innovations in computer technologies. Although dozens of BFT protocols have been introduced in the previous decade, their adoption by practitioners sounds disappointing. To some extant, this indicates that existing protocols are, perhaps, not yet too convincing or satisfactory. The problem is that researchers are still trying to establish 'the best protocol' using traditional methods, e.g., through designing new protocols. However, theoretical and experimental analyses demonstrate that it is hard to achieve one-size-fits-all BFT protocols. Indeed, we believe that looking for smarter tactics like 'fasten fragile sticks with a rope to achieve a solid stick' is necessary to circumvent the issue.

In this thesis, we introduce the first BFT selection model and algorithm that automate and simplify the election process of the 'preferred' BFT protocol among a set of candidate ones. The selection mechanism operates in three modes: Static, Dynamic, and Heuristic. For the two latter modes, we present a novel BFT system, called Adapt, that reacts to any potential changes in the system conditions and switches dynamically between existing BFT protocols, i.e., seeking adaptation.

The Static mode allows BFT users to choose a single BFT protocol only once. This is quite useful in Web Services and Clouds where BFT can be sold as a service (and signed in the SLA contract). This mode is basically designed for systems that do not have too fluctuating states. In this mode, an evaluation process is in charge of matching the user preferences against the profiles of the nominated BFT protocols considering both: reliability, and performance. The elected protocol is the one that achieves the highest evaluation score. The mechanism is well automated via mathematical matrices, and produces selections that are reasonable and close to reality.

Some systems, however, may experience fluttering conditions, like variable contention or message payloads. In this case, the static mode will not be efficient since a chosen protocol might not fit the new conditions. The Dynamic mode solves this issue. Adapt combines a collection of BFT protocols and switches between them, thus, adapting to the changes of the underlying system state. Consequently, the 'preferred' protocol is always polled for each system state. This yields an optimal quality of service, i.e., reliability and performance. Adapt monitors the system state through its *Event System*, and uses a Support Vector Regression method to conduct run time predictions for the performance of the protocols (e.g., throughput, latency, etc). Adapt also operates in a Heuristic mode. Using predefined heuristics, this mode optimizes user preferences to improve the selection process.

The evaluation of our approach shows that selecting the 'preferred' protocol is automated and close to reality in the static mode. In the Dynamic mode, Adapt always achieves the optimal performance among available protocols. The evaluation demonstrates that the overall system performance can be improved significantly too.

Other cases explore that it is not always worthy to switch between protocols. This is made possible through conducting predictions with high accuracy, that can reach more than 98% in many cases. Finally, the thesis shows that Adapt can be smarter through using heursitics.

Keywords. Fault tolerance, Byzantine fault tolerance, fail-stop, benign faults, paxos, BFT selection, adaptable BFT, dynamic switching, BFT prediction, BFT support vector machines, BFT analysis, abortable BFT, Abstract, Re-Abstract, Byzantine resilient OpenLDAP, heuristic BFT, one-size-fits-all BFT, preferred BFT protocol, BFT quality of service, BFT clouds.

Acknowledgments

I would like to express my grateful thanks to AZM association who supported this work; this thesis could not have been accomplished without AZM support.

My special thanks go to my supervisors Jean-Paul Bahsoun and Kablan Barbar for giving me the opportunity to work with them on this research and for their directions. Also, I am grateful to them for their trust and confidence; they gave me a large margin of freedom that was essential to gain experience on how to resolve problems from a to z.

I do not know how to thank Prof. Rachid Guerraoui on both professional and personal levels. Rachid proposed the subject of my thesis and stood with me until graduation. I thank him for his valuable directions and advices. He taught me how to understand problems, deal with them, and then present them in a paper. I would briefly say: he taught me how do do research. My deep gratitude to Rachid for his trust and encouragement in many situations. I thank him for inviting me twice to his laboratory and gave me the opportunity to work with his team at EPFL. On the other hand, Rachid is an interesting person who gave me great remarks and advices for my personal life.

I present my warm thanks to the jury of my PhD defense, the professors: Hermann De Meer, Miguel Correia, Jean-Marc Pierson, and Bilal Chebaro. Prof. Hermann tried his best to review my thesis being very busy; Miguel gave many interesting comments, advices, and proposals; Jean-Marc was very generous being the leader of my group; and finally, Bilal helped me in many administrative situations during my PhD.

I must not forget to thank my friend Robert Basmadjian for his help, encouragements, and advices. Robert is really a great person. Also, I appreciate the kindness of my current post-doctoral supervisor Sonia Ben Mochtar for her understanding while writing my thesis during work hours.

I would like to thank Rachid's team who I worked with at EPFL: Marco, Vivien, Maysam, and Nikola. I learned many useful techniques and tricks from them; they were very kind and collaborative.

My deepest gratitude goes to my parents, brothers, and sister. They supported me over the years, they were always seeking the best schools and universities for me to have a reputable education; and they were gladly waiting this moment.

All the remaining thanks and love go to my wife who stood with me in hard and happy moments. She reviewed all my papers and thesis; she listened to me when I needed to speak; she used to be happier than me in moments of success, and sadder than me in obstacles. I do not know how to thank her, words have gone ...

Preface

This thesis presents the research work done during my PhD under the supervision of Prof. Jean-Paul Bahsoun and Prof. Kablan Barbar. The thesis subject was proposed by Prof. Rachid Guerraoui who co-advised my work over the years of my research work. Rachid invited me twice to his laboratory at EPFL, Switzerland, where I spent six months as a research intern. I worked with his team: Marco Vukolić, Vivien Quéma, Maysam Yabandeh, and Nikola Knežević on many BFT topics: abortable BFT, Byzantine Resilient Directories, Obfuscated BFT, and BFT performance assessment using Queuing theory. A part of my work was published in international venues, and the remaining part is under submission. Hereafter, I list some of the published and under-submission papers:

[1] Ali Shoker and Jean-Paul Bahsoun. *Towards Byzantine Resilient Directories.* In the 11th IEEE International Symposium on Network Computing and Applications, IEEE Computer Society, Cambridge, MA, USA; August 2012.

[2] Ali Shoker and Jean-Paul Bahsoun. *Recover to Self: BFT Re-Abstract Family.* In the proceedings of the International Conference on Computer and Management (CAMAN), IEEE, March 2012.

[3] Jean-Paul Bahsoun, Rachid Guerraoui, Ali Shoker. *Can BFT be: Adapt-able?.* Under submission, 2013.

[4] Jean-Paul Bahsoun, Rachid Guerraoui, Ali Shoker. *BFT Selection.* Under submission, 2013.

[5] Jean-Paul Bahsoun, Rachid Guerraoui, Ali Shoker, and Maysam Yabandeh. *Obfuscated BFT.* Under submission, 2013.

Contents

Contents

Chapter 1

Introduction

Fault Tolerance is a fundamental requirement for reliable services. The achievements towards tolerating *benign faults* [34, 6, 24, 42, 34] (i.e., fail-stop or crash faults) are nowadays well mature, and thus are being deployed everywhere. However, studies that address *Byzantine faults* [35, 15] have not yet settled on a 'convincing' solution for practitioners. Consequently, research community is still probing for new solutions that are satisfactory in two aspects: reliability, and performance. Currently, this field has become crucial with the emergence of new threats due to the reliance on the Internet, and because of the evolution of computer technologies that yield new bugs and incompatibility issues. Though a lot of BFT solutions emerged recently to resolve this problem, it is still vague for practitioners to decide what is the 'best' protocol to choose.

Byzantine Fault Tolerance (BFT) is a technique used to leverage the resiliency of partially-synchronous [41, 23, 38] services against arbitrary faults, the so called *Byzantine* faults. BFT is often used on state machine replication (SMR) [47, 33]. To make such a service Byzantine resilient, it should be deployed as a *deterministic* state machine on multiple replicas and use a BFT protocol ([15, 2, 22, 32, 26, 50, 19]) to manage the communication between replicas and clients. BFT protocols tolerate Byzantine faults as long as up to one third of the replicas, and any client, are Byzantine [12, 15, 14].

Plenty of BFT protocols have been introduced recently. PBFT [15, 14] is considered the bedrock of practical BFT protocols. It maintains fault tolerance in the presence of Byzantine faults. Unfortunately, the performance of PBFT is low. Afterwards, the major concern of research community was to boost up the performance of PBFT even if robustness is a bit compromised; thus, they used speculation [32, 2, 22, 26] for this sake. The argument is that stable services are often reliable, but they mostly suffer from transient failures or attacks. Indeed, the perseverance of researchers then yielded many BFT protocols with better reliability and performance, however, this needed some sacrifice in other properties. Any new upcoming protocol used to fill some of the gaps of its predecessors and, unfortunately, introduced other new issues.

Nowadays, various BFT protocols are present (e.g., [15, 32, 2, 22, 26, 1, 28, 4, 50, 19], etc). Each has its strength and either exhibits new weaknesses, or imposes new assumptions to guarantee the anticipated results of its design objectives. For instance, PBFT [15] operates in the presence of Byzantine nodes, but it suffers from low performance under some conditions as compared to speculative protocols. Q/U [2] and Quorum [26] exhibit the lowest latency and fault scalability, however only in contention-free services. Zyzzyva [32] and Chain [26] achieve a high throughput, but rather they suffer from expensive recovery. Ring [1] scales to a high number of clients under contention, but it exhibits a low performance. Aardvark [19], Prime [9], and Spinning [50] provide more robustness and tolerance to DoS attacks, but they either remained vulnerable to some attacks, or required more complex infrastructure and resources, etc. Abortable BFT protocols like Aliph [26, 27] proposed a modular approach to use existing BFT protocols and *switch* from one to another upon failure. However, for the moment, no dynamic switching policy appeared to maintain run time system adaptability to the changes in the underlying conditions.

The discrepancy in the properties and performance of BFT protocols leads to the belief that one-fits-all BFT protocols are hard to achieve. This confuses BFT users while seeking the preferred protocol to use. A bunch of important questions thus arise: (1) What is the 'preferred' BFT protocol? And first, what is meant by 'preferred'? Could the various properties of BFT protocols be combined together in a single system? Which protocol to choose if the system state is often uniform? Can we provide dynamic run time adaptation in systems with alternating states? This thesis tries to answer all these questions.

1.1 Contributions

This thesis introduces the first BFT selection mechanism. The mechanism aims at guiding BFT users to choose their 'preferred' protocol in three modes: Static, Dynamic, and Heuristic. The former can be used in Web Services [39, 40, 8] or clouds [10, 54, 7] which can sell BFT as a service. Users are allowed to choose their preferred protocol. Moreover, we developed a new method, for this mode, to conserve the performance of existing BFT protocols under recovery. For the Dynamic and Heuristic modes, we rather introduce a novel BFT system, called Adapt. This system provides an optimal quality of service (i.e., reliability and performance) through combining existing BFT protocols together, and then dispatching one of them each time according to a dynamic selection policy. Adapt uses experimental Support Vector Regression (SVM) [11, 21, 49] forecast to assess the performance of different protocols in run time. Occasionally, Adapt uses some heuristics to boost up performance.

1.1.1 No one-size-fits-all Analysis

Our first contribution is to make an extensive study for the behavior of BFT protocols and their properties. The aim is to strengthen the belief that one-fits-all protocols are almost impossible to achieve in BFT. Therefore, we conducted three sorts of analysis. In the former, we describe the various characteristics of different existing protocols, showing that none of them could comprise all these characteristics. Then, we present a traditional theoretical analysis based on the basic design aspects of the protocols, e.g., message exchange pattern and number of authentications needed; but this time with a wider coverage and discussion. This analysis shows again that even theoretically BFT protocols achieve different performance (mainly throughput and latency). The third study is experimental. The experimental discussion is accompanied with a theoretical interpretation based on Queuing theory [25, 3]. Upon addressing different cases, we ended up with the same conclusion that: no BFT protocol is dominant in all cases.

1.1.2 BFT Quality of Service

Then we introduce a solution for the above problem. We describe, for the first time, a BFT selection model and algorithm used to cull the preferred protocol among a collection of protocols. Our definition to the ‘preferred’ protocol is: the most suitable protocol that matches the user preferences. Our approach is based on mathematical formulas to automate the selection process. We discuss all the parameters of the formulas and how to use them. In addition, we conduct many examples to evaluate the mechanism, and to convey that the output is reasonable and matches reality. The model operates in three modes: Static, Dynamic, and Heuristic. We explore these modes in the following.

1.1.2.1 Static Mode

In the static mode, the user should be able to choose a protocol that fits his requirement the most. This is quite useful with the revolution of Web Services and Cloud systems. To describe the method, we give this example: consider a service provider, say a cloud system, that sells IaaS, PaaS, and SaaS [10, 54, 7] services in a timely basis. The cloud vendor can provide an additional service (to be revealed in the SLA [7] contract) to leverage the service resiliency of the user’s service against Byzantine faults. The cloud vendor also provides a collection of existing BFT protocols that it supports, and uses our selection approach to help the users decide which protocol to choose. Protocols are usually accompanied with predefined profiles (i.e., a set of Key Quality Indicators KQIs) and users provide their preferences. KQIs are composed of Key Characteristics Indicators (KCIs) that define the properties of the protocol, and Key Performance Indicators (KPIs) that rank its performance. Selection occurs by running an evaluation phase. The evaluation occurs by matching the profiles of the protocols against the user preferences in an automated process. The protocol that achieves the highest evaluation score is considered the preferred protocol with respect to this very user. The service provider then assigns the suitable resources according to the user choice. In the static mode, the user is allowed to choose a single protocol only once, i.e., at the instant of system deployment (unless the terms of the SLA contract change). We conducted many concrete examples to evaluate this mode. The results show that the selection output is reasonable and exactly answers the demands of the users.

1.1.2.2 Re-Abstract Family

Since selection in the static mode is allowed only once, then according to the user, it is trivial to obtain (constantly) the anticipated output. In most current BFT protocols, this is not not possible. In fact, most exiting protocols suffer from drop in their performance upon failure detection, i.e., during the recovery phase. Experience showed that PBFT performs better than other protocols in the presence failures [32]; though, it is known of its poor performance in the trivial phase (i.e., with no failures). We introduce a new technique to maintain the same performance of existing protocols after recovery (e.g., until the faulty replica is repaired). We present the BFT Re-Abstract family [4] for this sake, based on the *abortable BFT* [27, 26] approach. Any Re-Abstract member runs the original Abstract (i.e., an existing BFT protocol) on $3f+1$ Active replicas, and requires f Passive replicas that are idle backups. Upon failure detection, Abstract recovers to itself again, but this time on a new Active set of replicas. The challenge is to evict the *Suspicious* replicas (Byzantine or just slow) out of the Active set and to replace them with *correct* ones (i.e., not Byzantine) from the Passive set. Abstract then returns to operate normally exhibiting its best-case performance. The evaluation of Re-Abstract demonstrates that the gain in throughput is sometimes up to twice that of Abstract itself while recovery. Also we show that Re-Abstract is very useful when the periods of failures in the system are long. Otherwise, if the failure is transient, the repair time of the faulty replica will be short (e.g., less than one second), and thus switching to Abstract itself will be worthless.

1.1.2.3 Dynamic Mode: Adapt System

The static mode works well in systems that are usually steady, i.e., the system state is not too fluctuating. However, if the system encounters fluttering conditions, the static mode will not be efficient. The reason is that, in the static mode, only one protocol is allowed to be run; what if this protocol does not fit the new conditions? For instance, running Quorum in systems that are deprived from contention is very beneficial, but what about its performance if the system had some periods of contention? The situation is more complicated if multiple impact factors are affecting the system state. The Dynamic state solves this issue.

For the Dynamic mode, we develop a novel BFT system, called Adapt, that can adapt dynamically to the changes of the underlying system state. Adapt ensures the optimal quality (reliability and performance) among existing protocols. It is designed for systems that experience fluttering states. Adapt can be applied on traditional platforms, Web Services, and Clouds.

Adapt is composed of three sub-systems: BFT Subsystem (BFTS), Event Subsystem (ES), and Quality Control Subsystem (QCS). BFTS is a pool of existing (and perhaps new) BFT protocols, it represents the BFT libraries (e.g., [15, 32, 2, 22, 26, 1, 28, 4, 50, 19], etc) and the *abortability* libraries ([27, 26]). ES monitors the system, and detects any potential changes in the system state that is represented by a collection of *Impact Factors* (e.g., number of clients, request size, data loss, etc). Upon detecting a *significant* change, ES sends an *event* to the *QCS*, thus triggering an evaluation phase. QCS maintains an optimal system performance by running the 'preferred' protocol in the next system state (i.e., under the new system conditions). QCS evaluates the BFT protocols by calculating their *Evaluation Score* (E); the protocol with the higher score is launched in the next phase. Evaluations are computed according to the selection model described above, but dynamically this time. In the Dynamic mode, the profiles of the protocols are not totally predefined; however, some of them are computed dynamically. QCS predicts the values of the Key Performance Indicators (KPIs) of each protocol, at run time, through using a well-known *Machine Learning* technique called *Support Vector Machines for Regression* (SVR) [21, 11]. Switching to a new protocol takes place only if it is worthy to switch with some chosen threshold.

In the evaluation of Adapt, we conduct concrete experiments on Redis [46] and OpenLDAP [58, 44, 43] applications. The experiments show that the prediction accuracy of throughput ranges between 95% and 98%. Then, we discuss how the impact factors (e.g., message payloads and number of clients) affect prediction accuracy. Also, in the evaluation we illustrate why this prediction accuracy is enough for Adapt to decide what is the preferred protocol, and whether switching is worthy or not. Our experiments demonstrate that Adapt can always choose the preferred protocol through applying the Dynamic mode.

1.1.2.4 Heuristic Mode

Although the Dynamic mode can achieve good results, however, this is limited to the user preferences. Since BFT service administrators usually have more knowledge than users (about BFT protocols), it is good to define some heuristics to optimize the choice of the users. This can be quite useful sometimes. For instance, if the user requires a high throughput, then Adapt will choose the preferred protocol; however, when there is no contention on the system, experience shows that latency becomes more important than throughput. Another example is when the system is overloaded with requests due to the high number of clients accessing the system, tolerating a high number of clients becomes more crucial than high throughput (though they do not contradict). We show how to automate such heuristics in the selection process using the same mathematical formulas introduced in the selection model. Then we describe how to use the Heuristic mode in Adapt. The evaluation of the Heuristic mode explains how it can improve the choice of the user to achieve the better quality of service. Defining a complete list of heuristics is a future plan.

1.2 Thesis Organization

The rest of the thesis is organized as follows.

In Chapter 2, we start by defining Byzantine fault tolerance. Then, we demonstrate why tolerating crash faults in not enough. Then we present some of the common BFT concepts and techniques. Well-known state of the art BFT protocols (used in this thesis) are then recalled concisely.

Chapter 3 motivates this thesis by illustrating why one-size-fits-all protocols are hard to achieve in BFT. It conveys three types of analysis. The first one considers the different characteristics of existing protocols. The second analyzes the performance of the protocols considering their communication patterns. The third analysis then discusses experimentally the discrepancy of performance among BFT protocols, accompanied with a justification based on Queuing theory. Just before introducing the third analysis, we describe in this chapter the system applications and settings used to conduct these experiments.

Our selection model is presented in chapter 4, preceded by some related work and terminology. The selection algorithm is then presented, and the mathematical selection formulas are thoroughly explained with many examples are presented. The chapter explores the three system modes: Static, Dynamic, and Heuristic. Then, we evaluate these modes, and we present a method to discover when switching is worthy.

Chapter 5 presents the Re-Abstract family. After introducing the approach, the chapter describes the design of the Re-Zlight, as a member of the family, explaining its two phases Speculative and Recovery. Re-Abstract evaluation is then conducted in the rest of the chapter, and terminated by showing how to use Re-Abstract in the Static mode.

Adapt is introduced in Chapter 6. The chapter starts by explaining why the Static mode is not enough and recalling some related work. The system architecture is then presented and followed by a description of the SVR prediction process. An interesting evaluation is then explored, followed by showing when switching is worthy. The Chapter ends by showing how to use the Heuristic mode in Adapt, and why this is important.

Finally, our concluding remarks and future work appear in Chapter 7.

Chapter 2

BFT Concepts and Protocols

In this chapter, we start by justifying the motivation behind Byzantine fault tolerance (BFT). Then, we recall some BFT concepts and properties, and we describe the BFT fault model. Afterwards, we explore a brief overview on the basic state of the art BFT protocols: PBFT, Zyzzyva, Q/U, HQ, Quorum, Chain, Ring, OBFT, etc. Recalling these protocols is important to understand the following chapters.

2.1 Why Byzantine Fault Tolerance?

Single client-server message exchange represents the simplest communication scheme in distributed computing; however, it suffers from harmful reliability issues. On the other hand, as the deployed services are becoming more open to the globe, a significant load on servers is increasingly imposed. In addition, introducing new computer technologies raised up the potential of failures in deployed services. These issues can easily compromise the reliability and performance of services. Replication is an essential technique used to maintain system reliability through having multiple backup servers (or redundant replicas). Replication aims to lift up system availability and performance while ensuring its correctness.

State machine replication (SMR) [33, 47] is basically used to make services tolerant to faults through achieving agreement among system replicas. Paxos [36] is a popular SMR protocol that handles *benign faults* (i.e., fail-stop or crash faults) [6, 34, 42, 37]. However, the increasing number of programming languages, platforms, and human attacks raised up another sort of arbitrary faults, called *Byzantine* [35, 14]. The difficulty of resolving such failures arise from the impossibility to assess the behavior of a Byzantine node, unlike fail-stop nodes that can be considered non-present. Byzantine Fault Tolerance (BFT) is the sort of replication that is in charge of maintaining consistency among system replicas in the presence of Byzantine faults.

The problem of Byzantine faults can be described in this example: consider a client that has a money credit x in a bank. The client withdraws a certain amount of money, say y. The client expects that the remaining credit in his account is (x-y). If the bank uses benign SMR replication, using Paxos for example, the system would return a receipt of (x-y) or it will fail. But what if it returns a receipt of (x-2y) for example? In this case, the system responded, but with a wrong response. BFT solves this problem by ensuring that any operation either finishes *correctly*, or stops.

2.2 Byzantine Fault Tolerance

2.2.1 Definition and Approach

Byzantine fault tolerance [35] (BFT) is a technique used to maintain resiliency against *Byzantine* (arbitrary) faults. Byzantine faults are those faults that impose, on services, an unpredictable behavior that is different to the purpose of their design; they include incorrect processing of a request, sending inconsistent responses, maliciously corrupting a state, etc. A BFT protocol is used to manipulate the communication among system replicas and clients assuming partial synchrony [41, 23, 38]. Typical BFT protocols require at least $3f+1$ replicas to ensure consistency among system replicas, where at most f replicas can be Byzantine [35, 15, 12]. Some protocols like Q/U [2] requires at least $5f+1$. In addition, trusted components can be used with certain synchrony assumptions to reduce the minimum number of replicas to $2f+1$, e.g., in [20, 17, 51].

BFT consists of replicating a service over a group of servers (called replicas). Each replica maintains a set of state variables, which are accessed by Read/Modify operations. Operations are executed *atomically* in a deterministic way, i.e., starting from the same initial state on all of the replicas, and executing the same operation on any replica should generate the same final state. BFT protocols should satisfy two properties:

- safety: all processes of correct replicas execute the same requests in the same order;
- liveness: all correct requests from the client processes are eventually executed.

BFT protocols are designed to maintain the above two properties. In such protocols, correct clients issue requests to be executed on replicas; these requests are disseminated among replicas according to a certain message exchange pattern (i.e., seeking agreement), and thus they are totally ordered. Ordering is often done by a specific *primary* replica (except for client-based protocols). At the end, one replica (or more) sends back the replies to the client. Replicas and clients maintain request histories to recover under failure. In BFT, the detection of Byzantine faults and recovery takes place with the help of clients (they are forced to contribute in this process in order to have their requests served).

2.2.2 Authentication

To tolerate malicious attacks, exchanged messages in BFT protocols are authenticated via some cryptographic techniques, such as Public Key Cryptography (PKC) or Message Authentication Code (MAC). PKC could make the BFT protocols much simpler since it is verifiable even after the message is forwarded multiple times, but it is much slower than MAC. The throughput can be bounded by the number of MAC or PKC operations per request performed by the bottleneck replica; because the sender has to authenticate a message for each destination. The node that sends more messages to the other nodes, therefore, has to do more MAC or PKC operations as well.

2.2.3 Understanding the $3f+1$ Bound

To understand why at least $3f+1$ replicas are needed to tolerate f Byzantine replicas, we make a short comparison with the case of crash faults where only $2f+1$ replicas are needed[34].

Starting with crash faults (e.g., Paxos), consider a replicated service that maintains mutual exclusion on Read/Write operations. Suppose the number of replicas is n, and assume that up to f replica failures can occur during a *short* window of time. Now, if a Write operation is performed, then the client waits until it receives n-f replies only to maintain liveness since f replicas might not respond at all (system is partially-synchronous [41, 23, 38]), thus it should not wait more. When another Read request is issued, the client waits for n-f replies too, but probably from another set of replicas. Thus the common set of replicas between the Read and Write quorums is n-$2f$. For the service to be correct, Read and Write quorums must intersect in at least one replica since in fail-stop faults we are sure that this replica is not faulty (otherwise it should be crashed); hence n-$2f$>0. Consequently for f faulty replicas, at least $2f$+1 replies are needed to maintain safety and liveness.

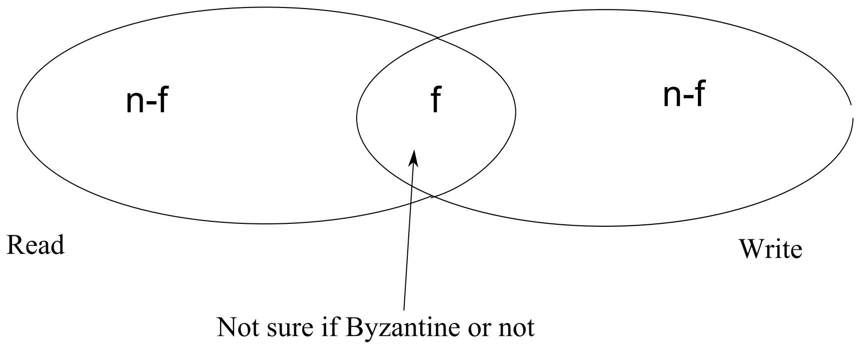


Figure 2.1: Intersection of Read/Write quorums in BFT.

However, in BFT case (see Figure 2.1), it is hard to decide whether the f replicas that did not respond are Byzantine or just slow. If they were just slow, then there might be f Byzantine replicas among the first operation (Read or Write) quorum. Thus, among the intersection n-$2f$, there might be f Byzantine replicas that only responded either in the Read operation or in the Write operation since Byzantine replicas can arbitrarily or selectively respond. Therefore, for the service to be correct, the intersection should include at least one correct replica in addition to the f suspicious replicas (that might be Byzantine), hence n-$2f$>f; in other words, n>$3f$. Therefore, at least $3f$+1 replicas are required to maintain safety and liveness assuming that at most f replicas are Byzantine.

2.3 BFT System Model

BFT fault model [15] assumes a message-passing distributed system using a fully connected network among nodes: clients and servers. The network may (not infinitely) fail to deliver, corrupt, delay, or reorder messages. Faulty replicas and clients may either behave arbitrarily, i.e., in a different way to their designed purposes, or they just crash (*benign* faults). A strong adversary coordinates faulty replicas to compromise the replicated service. However, we assume that the adversary cannot break cryptographic techniques like: collision-resistant hashes, encryption, and signatures. Liveness, however, is only guaranteed when the system is *eventually* synchronous.

2.4 BFT Protocols

2.4.1 PBFT

PBFT [15] is the seminal practical BFT protocol. It is considered a robust protocol (it works under failure); however, in most cases, it exhibits a low performance with respect to that of *speculative* protocols ([32, 2, 22, 26]). PBFT requires $3f+1$ replicas to ensure safety and progress assuming up to f replicas can be Byzantine. A *primary* replica is used to order the requests of the clients. A normal request operation is completed in three phases (see Figure 2.2): PRE-PREPARE, PREPARE, and COMMIT.

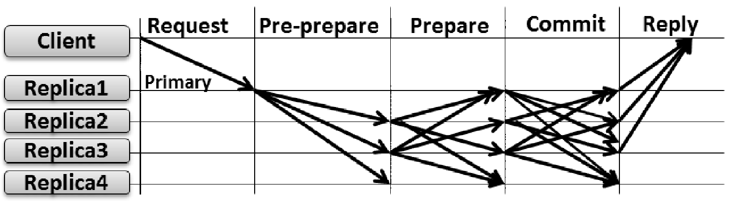


Figure 2.2: Message pattern of PBFT.

After the primary receives a request from a client, it appends a sequence number to this new request and broadcasts a PRE-PREPARE message to all of the replicas containing the ordered request. When a backup replica receives the PRE-PREPARE message, it acknowledges the message by broadcasting a new PREPARE message to all of the other replicas. As soon as any replica receives a quorum of $2f+1$ PREPARE messages, it promises to commit the request in its local history (ordered by the sequence number already appended to the request) by broadcasting a COMMIT message. Finally, when a replica receives a quorum of $2f+1$ COMMIT messages, it executes the request and replies to the client. The client commits the request if it receives $f+1$ matching replies; otherwise, the client retransmits the request.

Timers may expire on backup replicas while waiting for the primary to respond. In this case, a view-change phase is launched to assign a new primary. Consequently, each backup replica sends a VIEW-CHANGE request to all other replicas along with a sufficient number of CHECKPOINT and PRE-PREPARED messages to be used as a proof. When the new primary (usually the next replica) receives $2f$ valid VIEW-CHANGE messages, it multicasts a NEW-VIEW message, that comprises a set of PRE-PREPARED messages [1], i.e., a proof [15]. Backup replicas accept the new view after verifying the NEW-VIEW message from the primary; and the protocol resumes its normal operation.

2.4.2 Q/U

Q/U [2] is a Quorum-based BFT protocol that has better fault scalability and performance (especially latency) than PBFT; however, its performance drops under contention. The communication pattern of Q/U in the failure-free phase is simple (Figure 2.3): A client sends a request to a *preferred* quorum of replicas, and the replicas reply directly to the client. Q/U need no primary replica to order requests; instead, consistency is maintained through storing object versions upon each update operation, the history of up-to-date

[1]Describing this process is out of the scope of this thesis.

object versions is exchanged among replicas and clients to ensure that the final version of an object is always being queried.

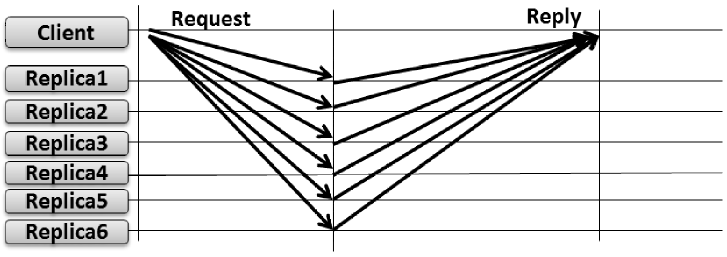


Figure 2.3: Message pattern of Q/U.

Q/U requires $5f + 1$ replicas to tolerate f Byzantine faults. Nevertheless, clients can contact a *preferred quorum* (of size $4f+1$) for optimum performance. This could result in outdated histories in some replicas, which induce the cost of a *synchronization* phase to the protocol. In this phase, the outdated replica requests the up-to-date history from $f+1$ other replicas (to ensure that the history is not manipulated by some faulty replicas).

Concurrent updates on an object may lead to different versions on different replicas, this is usually detected by the client that initiates a *repair* phase to recover from failures and brings the outdated copies of an object to the current version. Due to this expensive phase, Q/U is not scalable to large number of clients where the probability of concurrent updates is very high.

2.4.3 HQ

The Hybrid Quorum [22] (HQ) replication protocol is a quorum-based protocol that overcomes the limitations of Q/U (i.e., concurrency problems under contention) and PBFT (i.e., low latency and fault scalability). HQ requires only $3f+1$ replicas and combines quorum and agreement-based state machine replication techniques to provide scalable performance as f increases. In the absence of contention, HQ uses a new lightweight Byzantine quorum protocol in which Read operations require one round trip of communication between the client and the replicas, and Write operations require two round trips. When contention occurs, it uses an algorithm similar to PBFT to, efficiently, order the contending operations.

HQ exhibits a moderate performance between Q/U and PBFT in contention-free cases and a comparable fault scalability to Q/U. Under contention it is more robust than Q/U, but achieves a performance that is lower than that of PBFT.

2.4.4 Zyzzyva

Zyzzyva [32] is a BFT protocol that uses speculation to improve the performance of BFT state machine replication. Similar to PBFT, Zyzzyva requires $3f+1$ replicas to ensure safety and progress assuming that up to f replicas can be Byzantine. In the failure-free phase of Zyzzyva, the client sends the request to the *primary*, which assigns the request a sequence number and forwards it to the other replicas. The replicas immediately (i.e., speculatively) execute the request and send the reply to the client (Figure 2.4). In this way, Zyzzyva avoids the expensive agreement phase of PBFT that is needed to achieve total order of messages, prior to committing a request. The client completes a request in two cases: (1) if it received $3f+1$ *mutually-consistent* responses, or (2) if it received between $2f+1$ and $3f$ mutually-consistent replies.

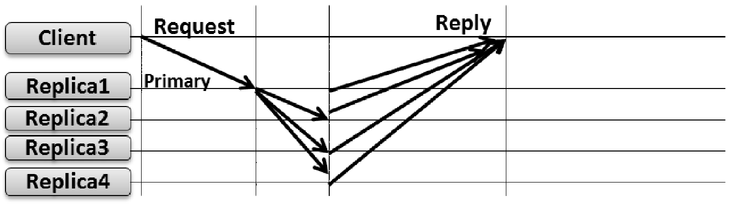


Figure 2.4: Message pattern of Zyzzyva.

In the latter case, the client sends a *commit certificate* to all replicas and waits for $2f+1$ SPEC-RESPONSE acknowledgments to complete the request; otherwise, the client retransmits the commit certificate to all replicas again which forward the request to the primary on the behalf of the client. When a replica receives an ORDER-REQ successfully from the primary, it confirms the response to the client; otherwise, it initiates a view change. In addition, the client may attain a proof of misbehavior (POM) against the primary (i.e., two different ORDER-REQ for the same request in the same view); in this case, the client forwards the POM message to all replicas to initiate a view change (in fact, the process is more complex; for more details see [32]).

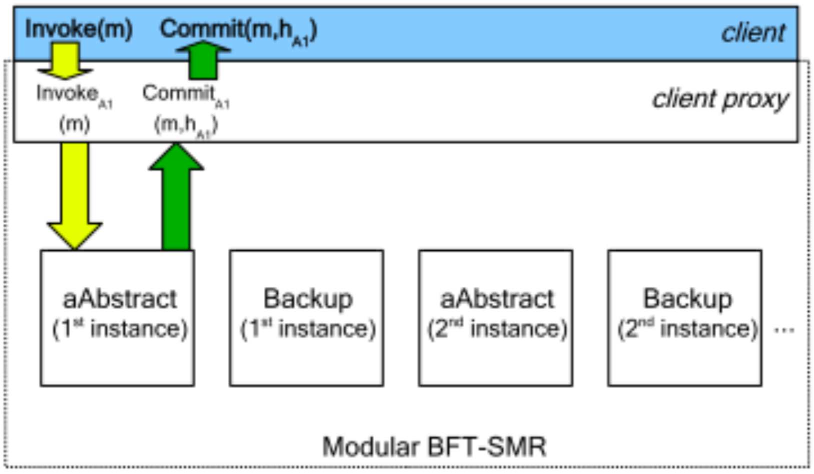
Although Zyzzyva exhibits a good performance and latency, its performance drops sharply under failure, even worst than PBFT by 15% [32]; the reason is refered to the complex recovery process described above.

2.4.5 Abortable BFT

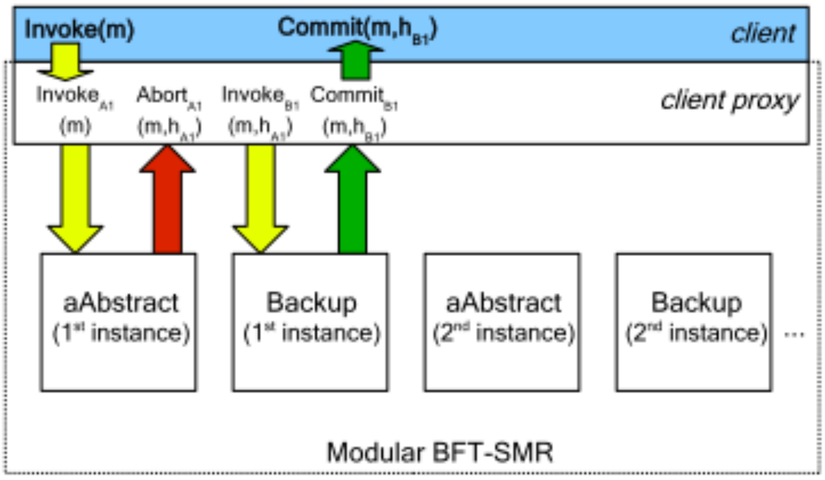
The notion of BFT abortability [27, 26] (i.e., Modular BFT-SMR) was introduced to reduce the complexity of design and implementation of BFT protocols. In our thesis, however, we adopt this approach to achieve optimal performance. An abortable BFT protocol (i.e., an *aAbstract*) that runs correctly if its *Non-Triviality* conditions are satisfied, can *abort* (i.e., stop running) at any time if these Non-Triviality conditions are no more satisfied; afterwards, no request will be serviced by the aborted protocol. A client can initiate abort if the currently running instance of BFT cannot safely progress anymore (e.g., because of contention or a failure), or the performance is unsatisfactory (e.g., because of a change in the payloads). The client then collects the *abort history* (i.e., the operations log) that is derived from the local histories of replicas. Then, the client is in charge of launching another protocol, called *Backup*, and initializing it with the abort history. The abort history can be exchanged among replicas and clients to stand as a proof against Byzantine replicas and clients. The whole process is decribed in details in Figure 2.5.

A new *Abstract* is initialized with an *abort history* that includes all the requests globally committed by the replicas in the previous BFT Abstract. The design of an abortable BFT protocol specifies a way to obtain the abort history, considering that (i) some replicas might be faulty and not responding; (ii) some of the responding replicas might be faulty and return invalid histories; and (iii) the histories in the replicas could be inconsistent because of contention or a Byzantine behavior.

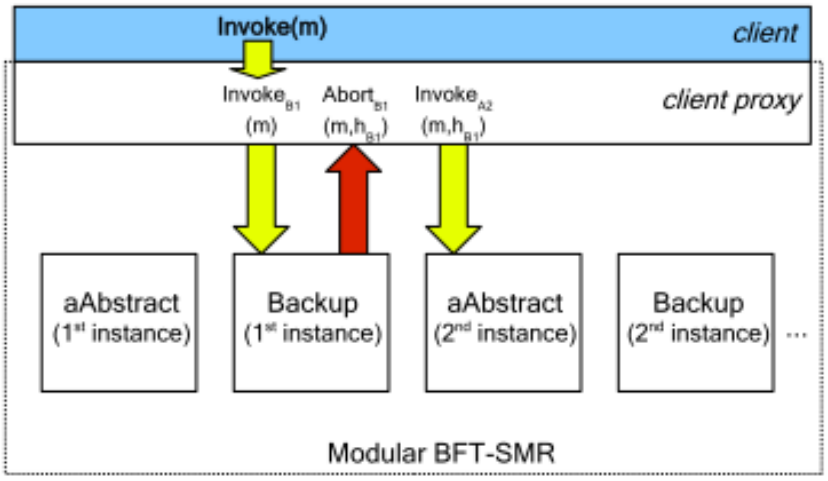
The modular behavior of abortability approach is very interesting since it allows using more than one protocol for a deployed service. However, this approach lacks a clear



(a) If aAbstract Non-Triviality conditions are satisfied, clients access only the aAbstract module.



(b) If aAbstract aborts, clients use the abort history as init history to switch to Backup. Backup is a powerful Abstract that guarantees to commit a certain number of requests.



(c) After a certain number of requests is committed within Backup, a client may be switched-back to try again (some) aAbstract.

Figure 2.5: Switching in abortable BFT.

dynamic switching policy to be used efficiently. In the history, only static switching of a predefined order of protocols was introduced. For instance, the authors in [26] introduce *Aliph*, a system of three Abstracts (Chain, Quorum, and Zlight) that can switch from one protocol to the next one whenever failures occur. Knowing that any existing BFT protocol can be made abortable, we exploit this feature in our thesis to provide new BFT solutions.

In the following we recall some abortable BFT protocols that will be addressed in the next chapters.

2.4.5.1 Quorum

Quorum [27] is a client-based BFT protocol that is designed to be abortable [27, 26]. Quorum achieves the theoretical minimum latency in contention-free cases. The protocol has the same message pattern as Q/U (Figure 2.6) in its trivial phase, i.e., in failure-free cases, however it requires $3f+1$ replicas to tolerate up to f Byzantine replicas (whereas Q/U requires $5f+1$). A client broadcasts the request to all replicas; each replica executes the request, updates its local history, and replies back to the client. The client completes the request if all the received replies match; otherwise, the client aborts Quorum, and launches another *stronger* backup protocol [27] (e.g., PBFT).

Although the performance of Quorum is very high (as compared to other protocols), it could not tolerate a large number of clients due to the concurrency problems under contention.

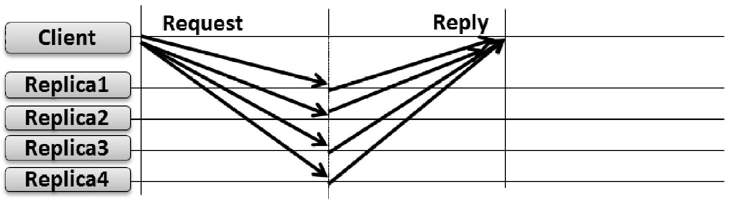


Figure 2.6: Message pattern of Quorum.

2.4.5.2 Chain

Chain [26, 27] is an abortable speculative protocol where system replicas are arranged in a chain fashion, starting by the Head replica and ending with the Tail replica. The communication pattern of chain is depicted in Figure 2.7. A client sends a request to the Head, which assigns a sequence number to the request. The head then forwards the request to the next replica in the chain. Each replica executes the request, appends it to its local history, and forwards the request to its successor until it reaches the Tail. Finally, the Tail replies to the client. The last $f+1$ replicas include the digest of their history in the forwarded request, to be sent to the client by the Tail. If these digests match, the client commits the request. Otherwise, the client resorts to a backup protocol to commit the request.

To make sure that no Byzantine replica is in the chain, each replica (except the Head and Tail) verifies $f+1$ MAC authenticators from its predecessor replicas and adds new $f+1$ MAC authenticators (corresponding to $f+1$ successor replicas) to the request. Although this technique imposes higher end-to-end delay, the throughput increases since the number

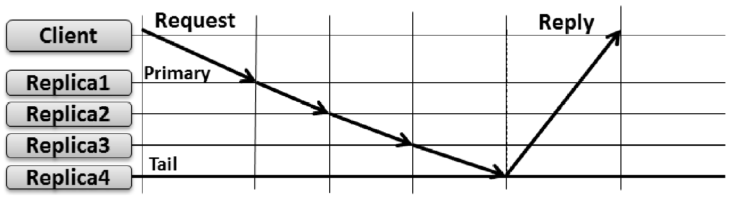


Figure 2.7: Message pattern of Chain.

of MAC operations performed by each replica is close to one, i.e., the theoretical lower bound.

2.4.5.3 Ring

Ring [1] is another speculative abortable BFT protocol that achieves a high throughput in the presence of large payloads and contention. Replicas are organized in a ring fashion where each replica has a *predecessor* and a *successor*. Clients can send requests to any replica and receive the reply from the predecessor of that replica. A *sequencer* replica is in charge of assigning sequence numbers to the requests that are forwarded from each replica to its successor (Figure 2.8). Any request moves in a complete turn through the ring to get assigned a sequence number by the sequencer, and then it continues in another round to execute the request on all replicas. This causes large delays in responses.

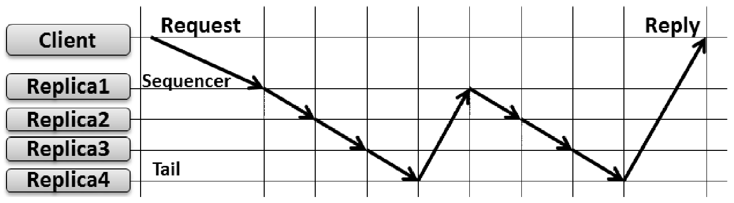


Figure 2.8: Message pattern of Ring.

Ring achieves a low throughput as compared to state of the art protocols; however, it dominates them under high contention, i.e., when the network becomes the bottleneck.

2.4.5.4 OBFT

OBFT [28] is client-based abortable BFT protocol that ensures obfuscation,i.e., replicas have no informtion about each other. The protocol dominates other client-based protocols (e.g., Q/U and Quorum) in client scalability. Furthermore, the protocol can completely ensure independence of failure among replicas; whereas, the other protocols require inter-replica interaction. For instance, the replicas in Q/U must communicate during the *Repair* phase, and Quorum recovers to a PBFT-like backup under failures which includes inter-replica communication too. Since OBFT is new to research community (it is one of our contributions), we describe it a bit more than other protocols. Further details can be found in [28].

OBFT requires $4f+1$ replicas to maintain safety and liveness; where the communication occurs among $3f+1$ *Active* replicas, and the extra f *Passive* replicas are used upon recovery. The communication pattern of OBFT is depicted in Figure 2.9. In OBFT, the client sends a request to the primary replica, the primary replica assigns the request a sequence number, executes it, and sends it back to the client. The client then forwards the ordered request to the remaining Active replicas which execute the request and reply back to the client. The client accepts the request if all the replies match (including that of the primary), otherwise, recovery is required. The recovery in OBFT is done by detecting the faulty replica among the Active replicas, and then replacing it with a correct replica from the Passive set. The mechanism is similar to the techniques described in Re-Abstract [4] in Chapter 5, thus we do not explain the method further in this section.

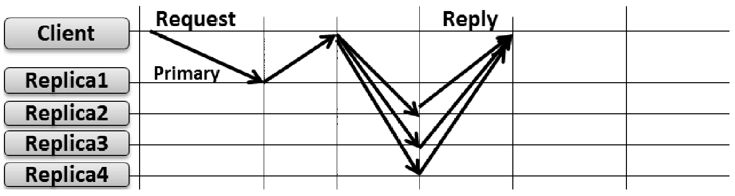


Figure 2.9: Message pattern of OBFT.

Since the client plays a crucial role in OBFT, the protocol do assume that clients can not be malicious, though they can fail by crashing. This assumption is mandatory to maintain obfuscation in any BFT protocol. In fact, if replicas do not communicate, a malicious client can violate consistency. The client sends two different requests to two distinct subsets of the replicas and behaves against each subset as if there was a single request.

2.4.6 Other BFT Protocols

A set of so-called *robust* BFT protocols have been recently introduced. For instance, Prime [9] uses some monitoring techniques to detect a slow primary and then initiates a view-change to maintain an acceptable performance under failures. Aardvark [19], on the other hand, uses similar techniques to Prime and attempts to use multiple backup NIC interfaces to avoid compromised replicas. Spinning [50] is a protocol that avoids attacks on the primary by changing the primary replica after every batch. Another protocol, i.e., Zyzzyvark [18], separates execution replicas from agreement replicas to maintain robustness under failures. These protocols aim at offering good throughput when faults occur, unlike the previous protocols that ameliorate the failure-free case. Although the concepts of our approach, in this thesis, are generic, i.e., any protocol can be included in the study, we do not discuss these protocols due to size limits (we are planning to include some of these protocols in future work).

Chapter 3

Does One-size-fits-all BFT Protocols exist?

In the previous chapter, we have shown that numerous BFT protocols have been designed to fill the gaps of their predecessors. In this chapter, we conduct three different analyses to show that no one BFT protocol dominates the others under all conditions. The first discusses the various characteristics of BFT protocols. The second studies the performance of BFT protocols considering their message exchange patterns. The third is an experimental analyses accompanied with a justification using Queuing theory. The experimental analysis is conducted on some applications that we describe briefly in this chapter too. We highlight some interesting cases where specific protocols are roughly superior to others. The aim of the three analyses, in this chapter, is to make a strong belief that one-size-fits-all protocols are very hard to achieve in BFT.

3.1 Discrepancy in the Properties of BFT Protocols

After introducing the seminal practical BFT protocol, i.e., PBFT [15], many protocols have been developed for the aim of improving its performance. This yielded a set of protocols that differ in their characteristics and performance. Then, any attempt for introducing a new protocol to resolve the issues of the predecessor protocols introduced other weaknesses in the new protocol; and thus, (1) no protocol was able to dominate the other protocols under all conditions, and (2) different protocols exhibited different design and robustness characteristics.

Table 3.1 depicts some of the different properties of state of the art BFT protocols. The columns and rows represent the protocols and their properties, respectively. In this table, we demonstrate that none of the protocols comprises all the characteristics of the others, and thus no single protocol could fit all users demands. A property of value 'Yes' means that the protocol has the corresponding property, whereas 'No' means that the protocols lack it. In the following we explain the studied properties:

- Speculative: this property tells whether the protocol is speculative or not. Speculative protocols are designed to boost up the performance in failure-free cases (they are optimistic towards the system state), however these protocols have three main limitations: (1) they exhibit a low performance under failures, (2) they are expensive when the network is instable (due to timeouts), and (3) they are complex to design and implement since they use some kind of 'undo' operations. In other words, speculative protocols achieve better performance and less robustness. The first row of the table shows that PBFT is the only non-speculative protocol among the others. This makes PBFT the most robust protocol in this comparison; whereas, the other protocols outperform PBFT in failure-free cases.

- Byzantine clients: a protocol that has this property tolerates Byzantine clients. According to the second row of Table 3.1, OBFT is the only protocol that does not tolerate Byzantine clients; on the contrary, it assumes that clients can not be malicious. As mentioned in the previous chapter, this assumption is mandatory to maintain obfuscation in BFT.

	PBFT	Zyzzyva	Q/U	HQ	Quorum	Ring	OBFT	Chain
Speculative	No	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**
Byzantine clients	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	No	**Yes**
Tolerates contention	**Yes**	**Yes**	No	**Yes**	No	**Yes**	**Yes**	**Yes**
Supports WAN	**Yes**	**Yes**	No	**Yes**	No	**Yes**	**Yes**	**Yes**
Self-Recovery	**Yes**	**Yes**	**Yes**	**Yes**	No	**Yes**	**Yes**	No
Obfuscated	No	No	**Yes**	No	**Yes**	No	**Yes**	No
Requires $\leq 3f+1$	**Yes**	**Yes**	No	**Yes**	**Yes**	**Yes**	No	**Yes**

Table 3.1: Some properties of different state of the art BFT protocols.

- Tolerates contention: this property represents the number of clients that are addressing the service simultaneously. Table 3.1 shows that Q/U and Quorum could not tolerate high contention. This is expected since these protocols are quorum based, and thus they do not utilize a primary replica to order requests. This may lead to different versions on different replicas and, consequently, recovery is needed. However, these protocols outperform the other protocols with few client due to their short messaging pattern that infers a low response time.

- Supports WAN: this property could not be ensured by protocols that use IP multicast since this feature is not yet supported in WANs. Consequently, the protocols: Q/U, and Quorum could not run on WANs in their current design (though it is not hard to use uni-cast instead, but this leads to drop in their performance). The other protocols in the table, however, avoids multi-cast, and thus they support WANs.

- Self-Recovery: different copies on different replica might occur under failures or under high contention, this is usually resolved through a recovery phase. Though most BFT protocols implements a recovery phase, there are some protocols, e.g., abortable protocols, that do not include a recovery phase; they launch a backup protocol similar to PBFT instead. Table 3.1 shows that Quorum and Chain do not include a recovery phase.

- Obfuscated: this feature ensures no inter-replica communication and that none of the replicas knows a single information about other replicas (even about there presence). This is important in BFT since independence of failure is required otherwise assuming f faults would be hard to achieve (since all replicas may fail similarly). Table 3.1 coveys that Q/U, Quorum, and OBFT are the only protocols that can be obfuscated. Note that we consider Q/U as obfuscated though it sometimes requires inter-replica interaction to repair.

- Requires $\leq 3f+1$: this property defines the protocols that requires at most $3f+1$ replica to maintain safety and liveness. In Table 3.1, Q/U and OBFT require $5f+1$ and $4f+1$, respectively.

Considering the above study, one can simply notice that none of the protocols includes all features; instead, a protocol has some features and lacks others. Note that considering more properties can show more discrepancy, but we confine our study to these properties since we think they are enough to clarify our idea.

3.2 Theoretical Performance Analysis

In addition to their properties, BFT protocols differ in performance too. Table 3.2 depicts some key measures that are used to evaluate the performance of the protocols analytically.

The values of row A correspond to the minimum number of replicas needed to maintain consistency if at most f replicas are faulty. This value gives an intuition about the crowdedness of exchanged messages. For instance, Q/U requires $5f+1$ replicas whereas most of the protocols need only $3f+1$. The impact of these extra $2f$ replicas appears under contention (failures are more likely to occur); where the network gets jammed with a huge number of exchanged messages which leads to a significant drop in performance. This does not appear in failure-free situations due to the use of *preferred quorums* [2]. Moreover, the extra replicas of Q/U and OBFT makes these protocols a bit more expensive than the other choices. We skip the explanation of row B as it has less impact on performance.

Row C represents the number of MAC operations performed on the bottleneck replica. This is usually used to assess the throughput of the protocol. PBFT needs the higher number of MAC operations among all existing protocol, consequently, PBFT exhibits the worst throughput among them. Quorum and OBFT require only 2 MAC operations per request (for verification and signing) which is reflected clearly in their high throughput.

	PBFT	Zyzzyva	Q/U	HQ	Quorum	Ring	OBFT	Chain
A	**3f+1**	**3f+1**	5f+1	**3f+1**	**3f+1**	**3f+1**	4f+1	**3f+1**
B	**2f+1**	**2f+1**	5f+1	3f+1	3f+1	3f+1	3f+1	3f+1
C	2+8f	2+3f	4f+2	4f+2	**2**	≈4+4f	**2**	f+2
D	4	3	**2**	4	**2**	≈7f+2	4	3f+2
E	3	2	**1**	3	**1**	2	**1**	**1**

Table 3.2: Analytic evaluation for the state of the art BFT protocols tolerating f faults using MACs for authentication, and assuming preferred optimization (without batching): A represents the number of replicas needed to tolerate f Byzantine replicas; B is similar to A but excluding witness and backup replicas; C represents the number of MAC operations on the bottleneck replica; D is the number of one-way latencies needed for each request; and E represents the number of send/to kernel calls on the bottleneck replica. Bold entries denote protocols with the lowest known cost.

As for Row D which represents the number of one-way delays of a request, Quorum and Q/U again achieve the theoretical lower bound 2. The response time of these protocols is very low as compared to other protocol. This leads to a high throughput, again, in case few client are issuing requests. Ring has a very long message pattern which impose a big response time (however Ring is designed for congested systems).

Finally, row E represents the send/to kernel calls on the bottleneck replica. This value has a similar effect to the value of C explained above. PBFT and HQ need 3 send/to calls on the primary replicas whereas other protocols requires either 1 or 2 calls.

Notice in Table 3.2 that the lower bounds (bold entries) are scattered among the protocols which could not drive a clear decision about what is the preferred protocol. For the first glance, Quorum seems to be the best choice, but our experience shows that it could not work under contention (we show this later). Another observation can be drawn from the table is that, in general, Zyzzyva and Chain represent a trade-off between the protocols.

Therefore, according to this discussion, one can notice that no protocol in dominant over the other protocols under all conditions.

3.3 Experimental Settings

In order to perform a concrete experimental comparison between state of the art BFT protocols, we first describe the setup settings and the configuration of the system, then we conduct the comparison in the following sections.

3.3.1 Hardware Setup

To evaluate our work, we conducted our experiments on three different applications that have different performance characteristics. The setup of the system is done on Emulab [55] testbed, where we used *Dell d710* machines – 64-bit Quad Core Xeon systems, with 2.4GHz processor, 12 GB of RAM, and 2 NICs. At Emulab, each machine in our experiments runs Ubuntu 8.04, with 2.6.24 32-bit kernel. Replicas are systematically running on their own, separate machine, while clients are collocated on a total of 20 machines. Moreover, in all experiments we use 4 replicas (which consists in tolerating f=1 fault). Finally, we use a topology where the replicas belong to one Gigabit-Ethernet LAN, and the clients communicate with replicas over a second Gigabit-Ethernet LAN. Sometimes we change this configuration due to comparison purposes; we indicate this explicitly when changed.

3.3.2 Applications

3.3.2.1 Benchmarks

First, we evaluate the protocols using a standard a/b microbenchmark used first by Castro and Liskov in [15]; where a and b corresponds to the request and the response size, respectively. The microbenchmark is simple, as it resembles a single memory value that is updated or read upon every request. Most of previous work was evaluated through changing the payload of the exchanged messages along with increasing the number of simultaneous clients addressing the service. In this thesis, we use this microbenchmark to assess the throughput, latency, peak throughput, and scalability that each protocol attains as a function of the message payload size.

3.3.2.2 Redis

Redis [46] is an open source, in-memory key-value store. It supports high level atomic server side operations like intersection, union, and difference between sets. For performance reasons, Redis holds the whole dataset in memory. This results in a small processing time (around 20μs); in fact, this is the purpose of experimenting on such an application. Persistence is realized via an append-only file (a journal) that is written as operations modifying the dataset in memory are processed. We use the BFT library to totally order all accesses modifying the state of the key-value store.

In our integration of Redis and BFT, we utilize $3f+1$ (or more) replicas instead of a single Redis server. Redis clients issue regular Redis requests, which are caught by our BFT library, encapsulated, and then forwarded to the replicas. On replicas, after the agreement, our BFT substrate takes the request ready for execution, and passes it to Redis. Results are then forwarded back to the client. We measured the performance of Redis using a part of the accompanying *redis-benchmark* [46], namely the *SET* command. This command writes a value in the key-value store. We changed the benchmark to send *SET* commands whose size is 128 B, with an 8 B random key.

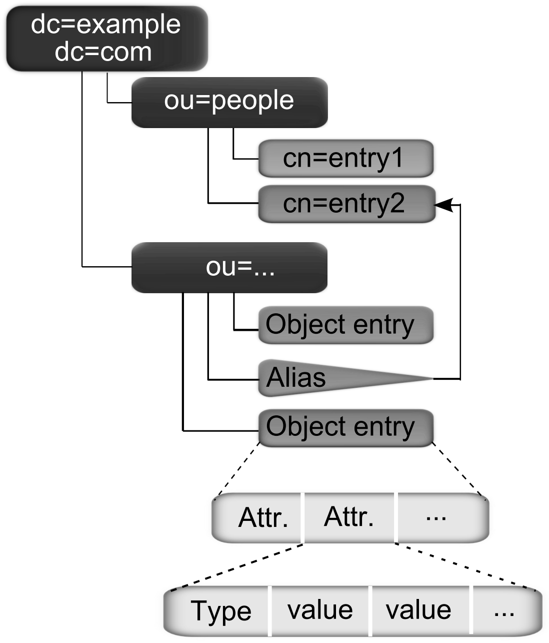


Figure 3.1: The structure of the *Directory Information Tree* DIT

3.3.2.3 OpenLDAP

Overview. Another application we use to compare BFT protocols is a *Directory* application called *OpenLDAP* [58]. The integration of BFT library with OpenLDAP is one of our contributions. We study OpenLDAP since very few works addressed database applications or applications of large execution time like OpenLDAP. Our integrated system, i.e., BFT-LDAP, yielded interesting observations: (1) applying BFT technology on services that provide a well-defined API, like OpenLDAP, is cheap and straightforward; and (2) the performance overhead of BFT-LDAP (due to replication) is fairly small (see [5] for more details).

LDAP [52] (*Lightweight Directory Access Protocol*) is a standard messaging protocol used to access (Read/Write) *Directory* information; where a *Directory* is a listing of information about objects arranged in some order [31]. The Directory information are organized in a hierarchical tree structure called *Directory Information Tree* (DIT). Figure 3.1 presents an example of DIT. The entries of a DIT are usually collections of attributes having values of different types, each entry has a globally-unique *Distinguished Name DN* (e.g., *entry*1.*people.example.com*).

Design. BFT-LDAP is designed as follows: in addition to the OpenLDAP server and the client application, three additional modules are built to integrate the BFT library onto OpenLDAP (see Figure 3.2): BFT Client Proxy, BFT Server Proxy, and LDAP Access Unit. The BFT client and the client application form the client tier, whereas, the other modules form the server tier, and hence they are installed on all replicas. The client application sends the requests to the BFT client proxy which forwards the request to the BFT server proxies after performing necessary message digests and encryptions. This communication phase takes place according to the specific message patterns of each BFT protocol. The BFT server proxy calls the LDAP Access Unit that interfaces the OpenLDAP server. After executing the operation on each OpenLDAP server, the reply is sent back through the same modules, however in the opposite sense, again considering the message patterns of the BFT protocol being used.

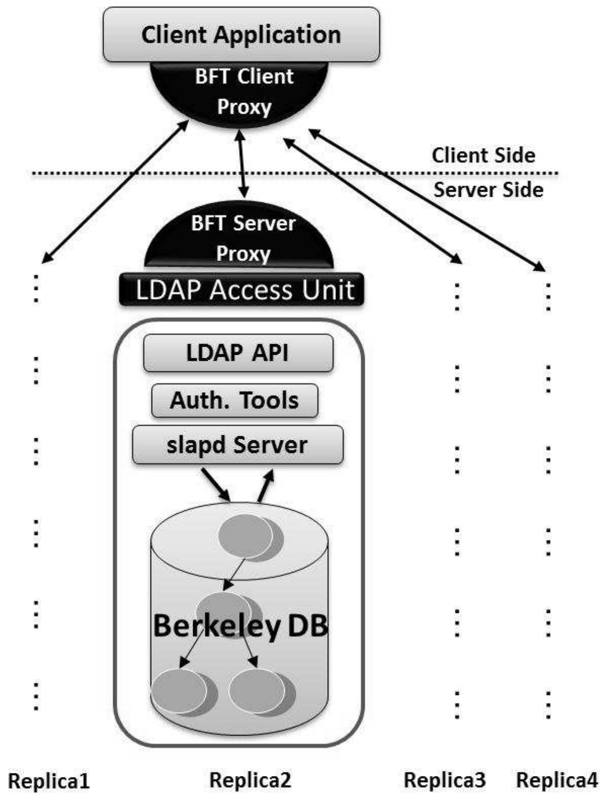


Figure 3.2: System architecture of BFT-LDAP for f=1.

The integration is similar for any BFT protocol since BFT libraries have a common interface that provides two basic methods: *issue_request()* that is called by clients, and *execute_request()* that is used by replicas to execute application-based requests (e.g., LDAP *Read/Update* operations). Speculative protocols require additional methods to handle ‘undo’ operations in their history. Our system was implemented in C/C++ code.

Installation and Configuration. Our experiments consider only *search* operations. We do not address *Update* operations for two reasons: first, search operations are the most frequent in Directories (more than 99% of total operations). Second, update operations may lead to replica inconsistency, and thus the BFT recovery phase is forced. This requires a *Transaction* mechanism (i.e., checkpoints) to retain one-copy semantics on all replicas. Unfortunately, transactions are not yet implemented in OpenLDAP [59] APIs, hence we leave this issue for future work.

OpenLDAP (v.2.4) is installed on all replicas with the Oracle Berkeley Database (BDB v.4). The databases are identical on all replicas and they are initialized with $100,000$ entries of 600 Bytes each. OpenLDAP and BDB are configured to provide a maximum performance. Thus, we allow any number of clients, any message size, indexing, and large cashing size (i.e., we use an in-memory database to avoid swapping).

3.4 Experimental Performance Analysis

The analysis conducted in Section 3.2 gives an intuition about the general performance of BFT protocols; however, this forecasting is not very accurate due to the various factors that affects the behavior of distributed systems on the ground. On the other hand, experimental analysis can give a deeper vision that is more accurate as the system conditions and factors change. This section discusses the behavior of BFT protocols under such conditions; and consequently supports our claim that one-size-fits-all protocols are hard to achieve in BFT.

3.4.1 Performance Queuing Model

Queuing theory [3] is often used to analyze the performance of different related processes in distributed systems. The main element in queuing analysis is a queue. Here, a queue may hold either a finite or an infinite number of requests, which arrive according to some distribution of arrival times. Figure 3.3 is the basic representation of the queuing model.

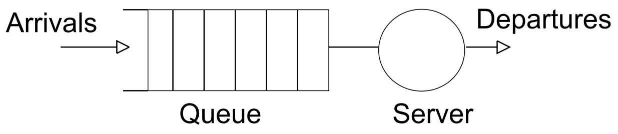


Figure 3.3: The basic queuing model scheme.

Using queuing theory, we formulate the time needed by a request to be handled and served. This time starts form the instant the client issues the request, passing by the replicas according to the communication pattern of the protocol used, and until the client receives the reply from replicas. Figure 3.4 explains the notations we use in the following discussion. In general, we consider only one replica in our analysis (usually the primary replica since it is the bottleneck) unless otherwise stated.

Denote by T_{cr}, T_{rc}, and T_{rr} the message delays of channels: client-replica, replica-client, and replica-replica, respectively. Then, the sum of these delays is denoted by:

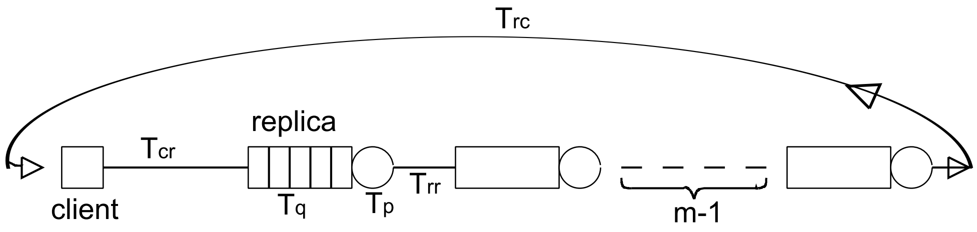


Figure 3.4: The time notations of a BFT system with multiple queues.

$$T_d = T_{cr} + m.T_{rr} + T_{rc} \tag{3.1}$$

The value of m represents the number replica-replica communications needed by a protocol. Notice that the value of m for the protocols Q/U and Quorum is zero.

Also let T_p be the processing time on a node which includes: (1) the execution time T_e, (2) the authentication time (verify and sign) T_a, (3) the send/to time (transmission) T_t, and (4) the handling time of other data structures T_h, hence:

$$T_p = T_e + T_a + T_t + T_h \tag{3.2}$$

In addition, a FIFO queuing time T_q is needed before each node to handle concurrent requests. Putting together, we get the total time needed by a request to be delivered as follows:

$$T_{total} = T_d + (m+1)T_p + T_q \tag{3.3}$$

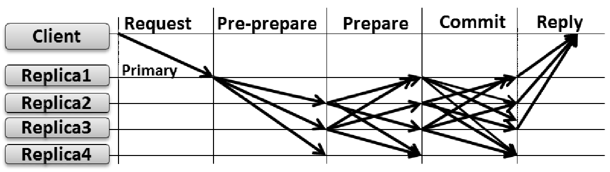
T_p is multiplied by m+1 since some protocols, like Chain, execute requests sequentially rather than concurrently (as in Quorum). The throughput of a BFT protocols is usually inversely, but not roughly, proportional to T_{total}.

Valuable observations can be drawn from the above equations. In the following, we discuss some of these observations that can help in interpreting the experiments we convey. The experiments cover a set of protocols: PBFT, Zyzyyva, Quorum, Chain, and Ring. We consider these protocols to achieve our goal showing that one-size-fits-all BFT protocols are hard to achieve. We do not consider other protocols for two reasons: (1) to avoid jamming this thesis with graphs and numbers, and most importantly (2) since other protocols have almost similar properties to those we consider in our study (e.g., Q/U similar to Quorum, HQ similar to Q/U and PBFT, etc). In the following, experiments are conducted using Redis as an application unless otherwise stated.

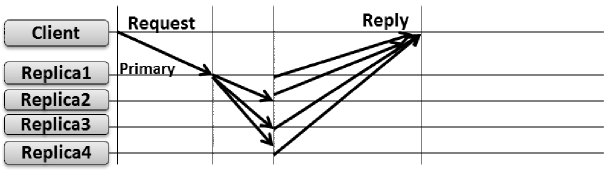
3.4.2 Single Client

Here we assume that no concurrent clients are issuing requests. In this case, T_q=0 since no queuing is needed and the client does not send the following request until it receives the reply of the previous one. Thus according to equation 3.3, the only playing factors are the T_d and T_p.

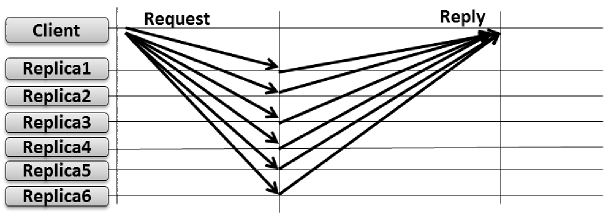
T_e is small. Suppose that the execution time T_e is small, e.g., using a/b microbenchmarks or Redis as an application. Then the authentication time T_a and the total channels delay T_d impose a high impact on the throughput of the system. Increasing T_d causes a drop in the throughput (and trivially increase in response time). Now, according to



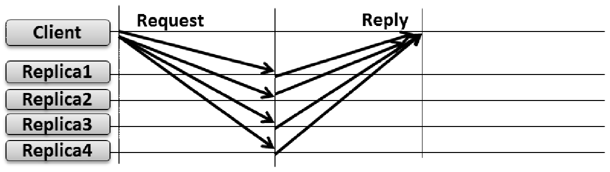
(a) PBFT



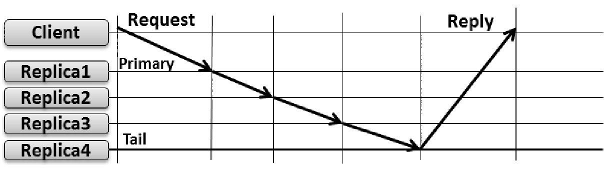
(b) Zyzyyva



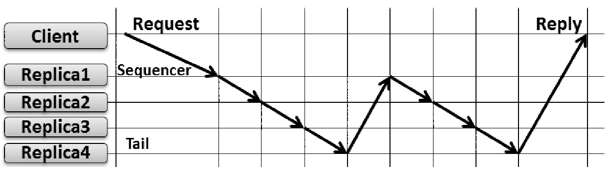
(c) Q/U



(d) Quorum



(e) Chain



(f) Ring

Figure 3.5: Message patterns of the state of the art BFT protocols; for $f = 1$.

equation 3.1, we have two possibilities: either (1) T_{rr} is large, e.g., replicas do not belong to same cluster; or (2) T_{rc} and T_{cr} are large, e.g., in WAN setting.

- In the former case (i.e., T_{rr} is large), the difference in performance among different protocols becomes large. This can be noticed clearly upon comparing Quorum and Q/U with the other protocols since T_{rr}=0 in both protocols; whereas, other protocols require additional inter-replica communication delays, i.e., $T_{rr} \neq$0. Figure 3.6 depicts the throughput and latency for a single client using two payload sizes 0KB (actually 64Bytes) and 1KB. The left (resp, right) histogram shows that Quorum

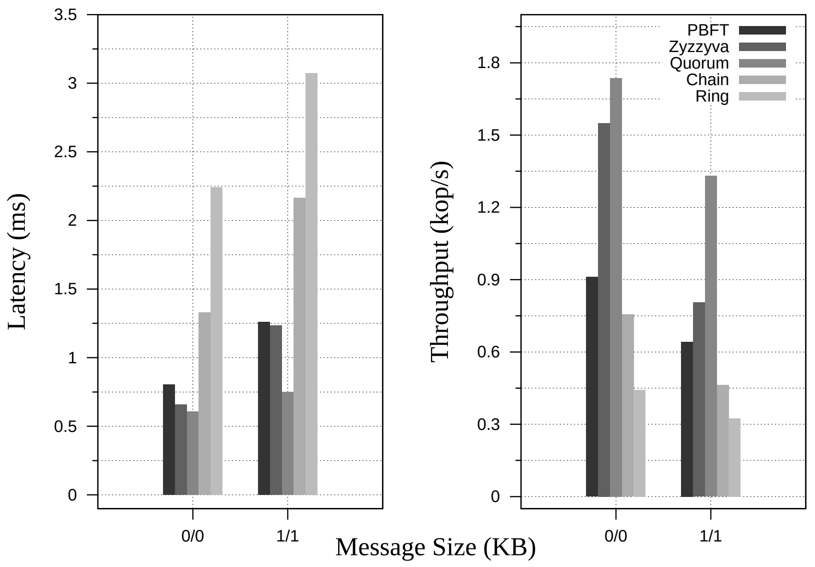


Figure 3.6: Latency and throughput of a single client using the 0/0 and 1/1 microbenchmarks.

achieves the lowest latency (resp, highest throughput) among protocols. PBFT and Zyzzyva require additional T_{rr} delay and thus their performance is less than that of Quorum (i.e., a higher latency and lower throughput). Ring and Chain, however, exhibit almost half the performance of the other protocols due to their extensive inter-replica communication as each replica needs to forward requests to the next replica according to the message exchange patterns in Figure 3.5. Notice that, as the message size increases, the latency increases (and throughput decreases) due to the additional processing needed by large messages (e.g., 1/1 benchmark).

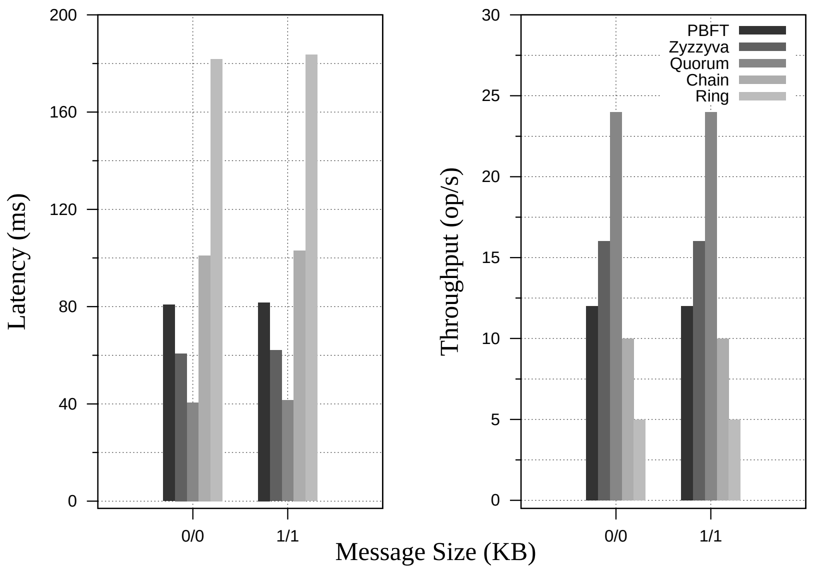


Figure 3.7: Latency and throughput of a single client and T_{rr} is large, using the 0/0 and 1/1 microbenchmarks.

In addition, the impact of T_{rr} on throughput can even be greater in the topology where different replicas are scattered over WAN, since the point-to-point (P2P) delay

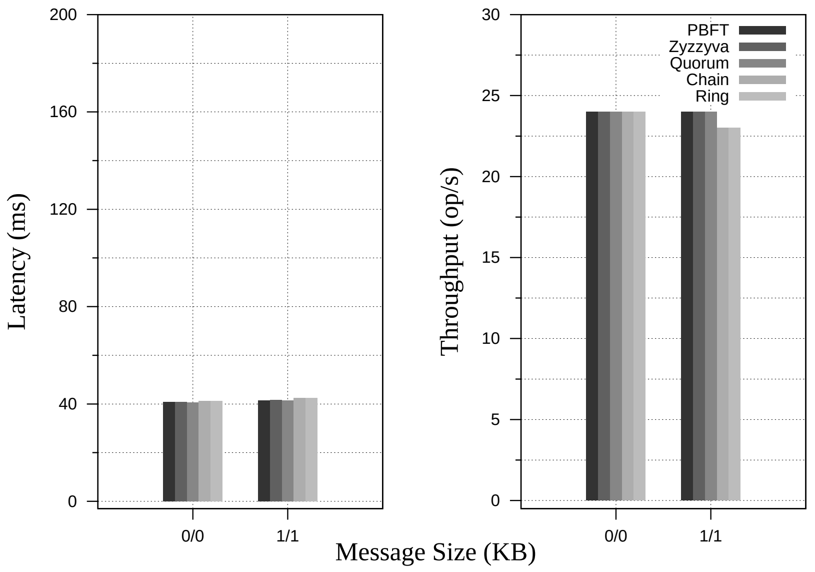


Figure 3.8: Latency and throughput of a single client in WAN setting, using the 0/0 and 1/1 microbenchmarks.

between replicas will be larger (usually greater than 10ms). This configuration is usually used to achieve independence of failures against power failures, earth quakes and disasters, etc. When the number of inter-replica communication m is large, as in Chain and Ring for instance (since each replica forwards the request to the next one), the response time will be very large causing a drop in the performance of the system in contention-free cases. The experimental results of Figure 3.7 confirms this analysis, and shows that Ring and Chain achieve the highest latency (and worst throughput) among other protocols in both 0K and 1K message payloads. Notice that, though PBFT also needs inter-replica communication similar to Chain (where f=1, see Figure 3.5), however, the latency of the former is less than that of Chain in contention-free environments. In fact, PBFT uses an optimization to send replies directly to the client (i.e., before the COMMIT phase) to improve its performance.

- The latter case where T_{rc} and T_{cr} are also large causes a drop in the throughput of the protocols. However, in this setting, the difference among protocols becomes negligible since these two factors dominate the effect of other protocol-related delays like T_a, T_t, and T_{rr} (which is small now, as replicas belong to the same LAN). Our experiments, as shown in Figure 3.8, depict the fact that protocols are roughly equivalent in performance with one client. In addition, the histograms convey that the impact of payload size is negligible, in such case, since T_{rc} and T_{cr} dominate the small processing delays of messages. A closer look at the same figure shows that Ring performs a bit worst than the other protocols; we refer this difference to the high inter-replica delays in Ring since it requires two message rounds around the *ring* to commit a request (see Figure 3.5).

In general, T_a and T_t have a similar effect to T_d. Protocols that require multiple MAC authentications and send/to calls to the kernel, i.e., T_a and T_t are large, would have a low throughput as compared to other protocols. PBFT is an example of such protocols since it needs 2+8f MAC operations and 3 send/to calls (see Table 3.2 in previous sections).

As the number of MAC operations and send/to calls decrease, e.g, as in Quorum and Q/U, the throughput of the system increases. These results do not appear in the above experiments (where a single client is addressing the service) since T_d dominates T_p in this configuration; however, with concurrent clients the difference can be noticed clearly.

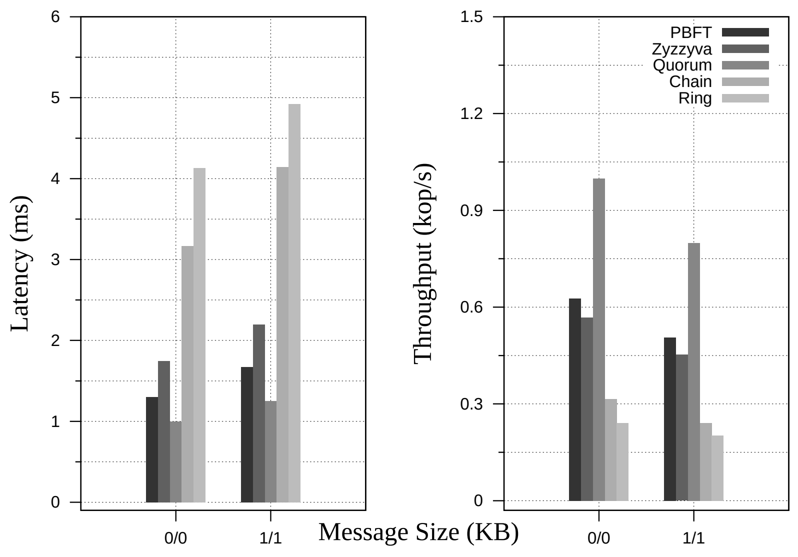


Figure 3.9: Latency and throughput of a single client and T_e is large, using the 0/0 and 1/1 microbenchmarks.

***T_e* is large.** If T_e is large (e.g., using OpenLDAP as an application), the throughput of the system drops down significantly. On the other hand, the difference in throughput among different protocols gets smaller since T_p dominates the effect of T_d; however, our experiments (depicted in Figure 3.9) show that this difference does not change significantly over the case where T_e is small above; with more clients instead, the difference appears clearly. In WAN settings, where P2P delay is greater than 10ms, the drop in performance will be negligible since T_d is usually larger than T_e (that does not exceed 1ms in most applications).

3.4.3 Concurrent Clients

Multiple clients could send simultaneous requests to replicas. If the requests are concurrent on a certain replica, it is most likely that they wait in a queue until the preceding requests are handled by the replica. This means that T_q does not equal to zero anymore, and thus, it affects the performance of the system.

The arrival and departure of requests to/from a queue (i.e., a replica) are often estimated by exponential distributions of an average arrival rate δ and average departure (or servicing) rate μ. Exponential distributions are convenient in such systems since requests arrive and depart randomly with a greater probability that they are close in time. According to Kendall's notation [3], the queuing system can be represented by M/M/k, where M is the exponential (Markovian) distribution and k is the number of servers. We focus our discussion on one replica i.e., k=1.

3.4.3.1 Execution Time is Small

When the execution time T_e is small, its impact on the processing time T_p becomes smaller, consequently, T_p fluctuates randomly; thus, the servicing rate of requests follows an exponential distribution of average rate μ=1/T_p. Supposing that the arrival of requests is also random, thus the arrival rate follows a Poisson distribution [1] of an average rate δ. Then, the average time a single request spends in the queue is as follows:

$$t_q = \frac{\delta}{\mu(\mu - \delta)} \tag{3.4}$$

Therefore, the total queuing time of all requests in the queue $T_q \neq 0$. In addition, μ=1/T_p, i.e., T_p is directly proportional to t_q, and hence, as the processing time T_p increases the queuing time T_q increases. Consequently, the expected raise in the throughout of the system (since more clients are issuing requests) decreases gradually until reaching some bottleneck (discussed in later sections). On the other hand, suppose that the number of waiting requests in the queue is n; then, the system finishes servicing the n requests as soon as the last (i.e., the n'th) request is served. This means that the systems will pay additional delay of $n \times t_q$. Therefore, as the number of queued requests increases, the total response time of the system (i.e., all requests) increases, and thus, the throughput curve keeps increasing (with smaller slope) until it reaches a plateau.

The impact of this drop is plotted on Figure 3.10. The figure shows the real throughput of Zyzyyva and its expected (unreal) throughput without the effect of queuing. In this latter case, the curve goes up linearly as the number of clients increases (note that the graph uses a logarithmic scale on x-axis to show the curves for few clients too). The figure conveys that Zyzzyva eventually reaches its maximal throughput (the plateau) early as compared to the case without queuing.

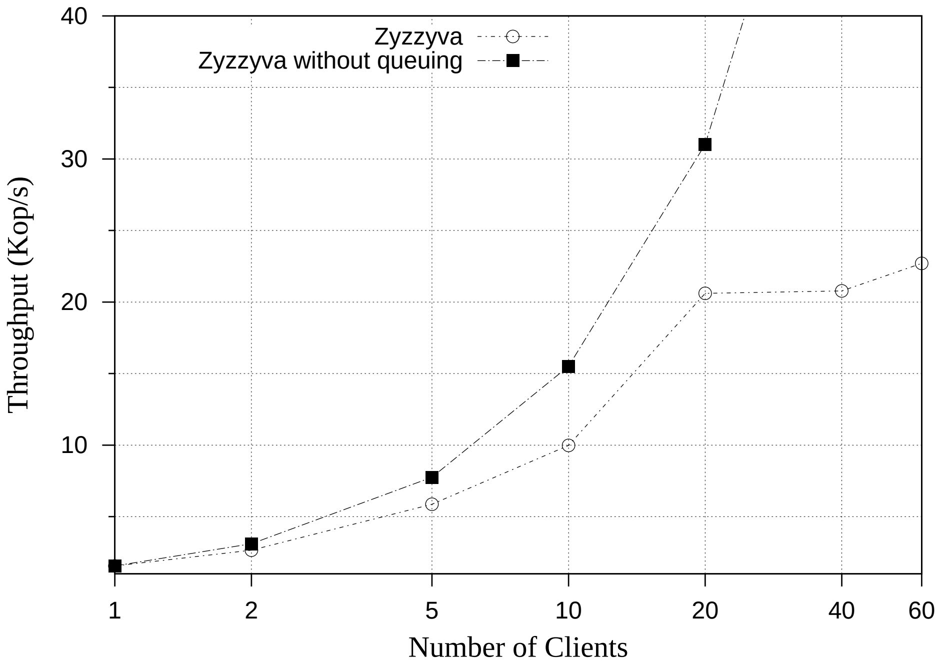


Figure 3.10: The expected throughput of Zyzyyva supposing no queuing happens under concurrency.

[1]Poisson is an exponential distribution that is often used for arrivals.

Notice that, since T_p increases as T_a and T_t increase, then the increase in throughput of the protocols varies as their T_a and T_t vary. Protocols that need more authentications and transmissions (e.g., PBFT) will have a slower increase in performance more than others. Figures 3.11 and 3.12 plot the throughput of different BFT protocols as the number of clients increases with both payload sizes 0K and 1K, respectively. We use the logarithmic scale on x-axis for clarity. The figures show that Quorum achieves the best throughput among its competitors, and that the difference between protocols increases gradually with the number of clients. Quorum achieves this since its queuing time is negligible as compared to other protocols that needs a minimal authentication and transmission delays (see the message patterns in Figure 3.5 above). In fact, the difference in throughput increases between all the protocols too due to the discrepancy of their T_a and T_t, and thus, the increase in the queuing time.

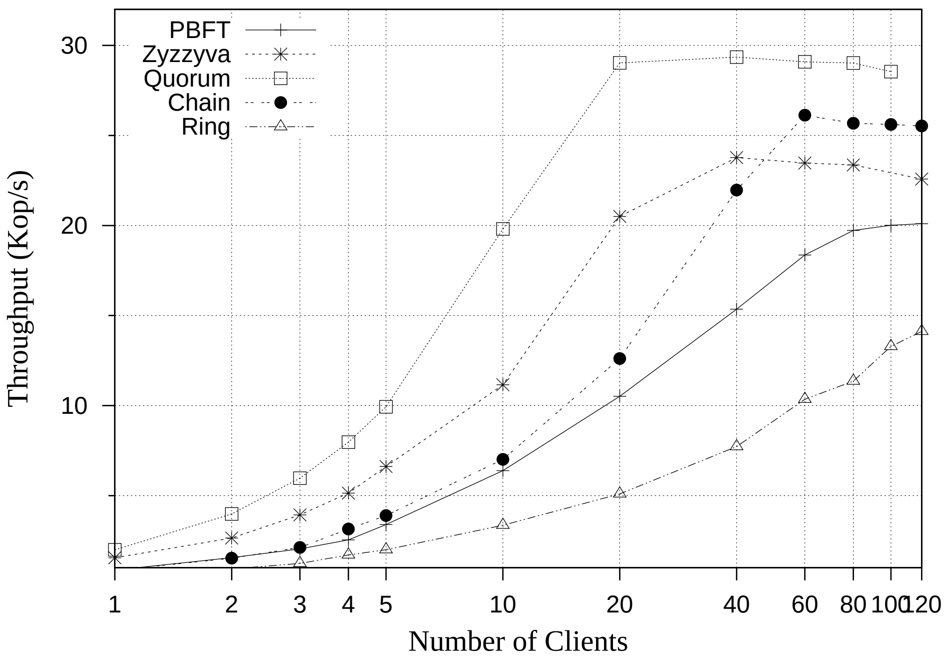


Figure 3.11: Throughput of concurrent clients, using the 0/0 microbenchmark.

Two more observations can be noticed in Figures 3.11 and 3.12 too. The first, is that Chain performs better than Zyzzyva with more than 50 clients. This can be explained by the fact that Chain requires fewer authentications (approaches to one authentication per request) as compared to Zyzzyva. This does not appear, however, with fewer clients since the effect of P2P delay (and hence T_d) is greater than that of the queuing time. The second observation is that Quorum could not operate with more than 40 clients in our case. This is expected since Quorum is designed to contention-free cases, hence it suffers from collisions and retransmissions under contention (this can be said about Q/U too since both protocols do not make use of a primary to order their requests).

3.4.3.2 Servicing Rate is Low

When the servicing rate μ is low, for instance upon using an application of high execution time T_e, e.g., OpenLDAP, then T_p will be high and almost constant since T_e masks the other factors (in equation 3.2). Thus μ=1/T_p becomes almost constant too. In this case,

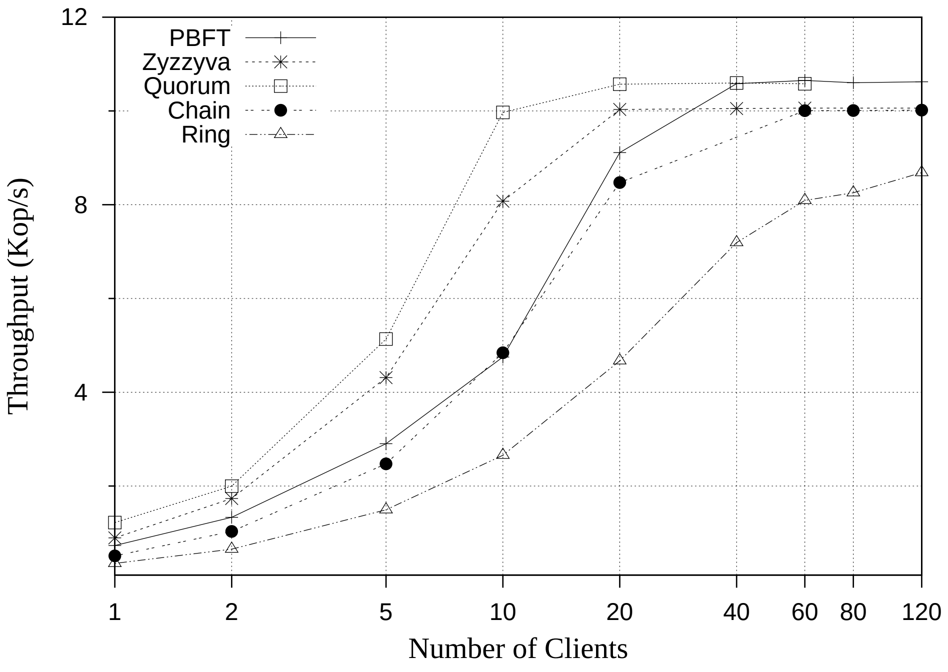


Figure 3.12: Throughput of concurrent clients, using the 1/1 microbenchmark.

the Deterministic distribution is used to model the servicing time (i.e., using M/D/1 Kendall's model) as there is a less randomness in the system [3]. Consequently, the queuing time is divided by 2:

$$t_q = \frac{\delta}{2\mu(\mu - \delta)}$$

As one can notice, the analysis is similar to the case of M/M/1 above with one exception: since T_e is very large, and thus T_p is almost constant for all the protocols, then the queuing time T_q will be equivalent for all protocols very soon. Therefore, the discrepancy in performance among different protocols will be negligible. This can be clearly noticed in Figures 3.13 and 3.14. The two figures depict the throughput of a set of BFT protocols while using OpenLDAP as an application for 0KB and 1KB payloads. Though a strong slope can been noticed in the curves with few clients; the curves continue horizontally with more than 5 clients in most protocols. In addition, the curves are almost confounded with each other beyond 5 clients which confirms the above analysis that when the execution time is large (e.g., approaches to $1ms$ in OpenLDAP) under contention, BFT protocols achieve a similar performance. The figures convey that Ring does not achieve this peak throughput. In fact, Ring achieves this very late (with more than 300 clients) due to its extensive two-round message pattern (see Figure 3.5). As for the drop in performance of PBFT in the figures with more than 40 clients, we refer this to a bug in the implementation (it used to work with more than 40 clients before having this issue).

3.4.3.3 Arrival Rate is High

In case the arrival rate of requests δ is high, i.e., the requests are almost instantaneous, then the ratio μ/δ approaches to zero, and the queuing time reaches its maximum. In fact, from equation 3.4 we get:

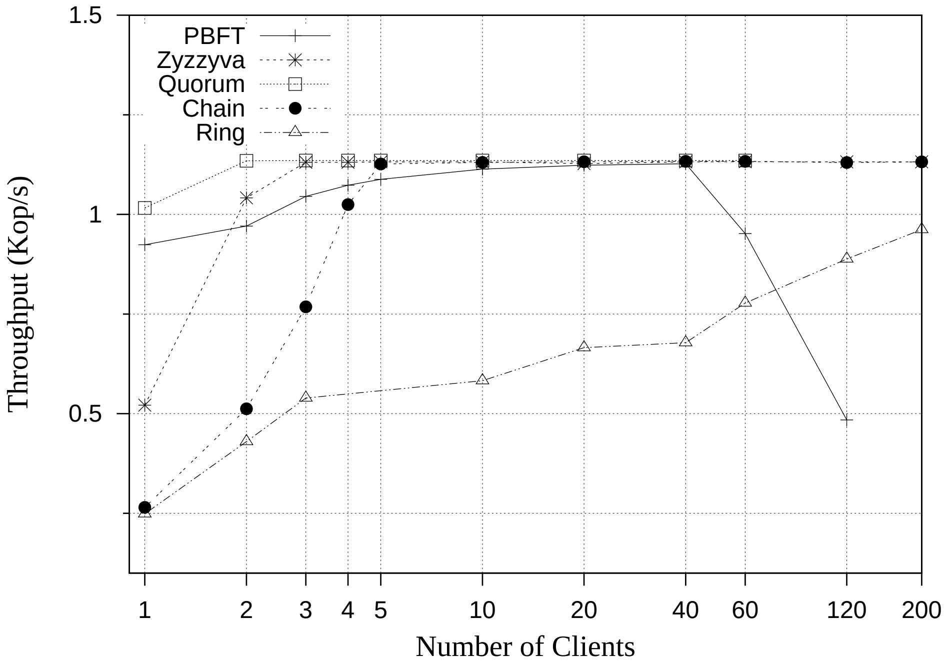


Figure 3.13: Throughput of concurrent clients where T_e is high, using the 0/0 microbenchmark.

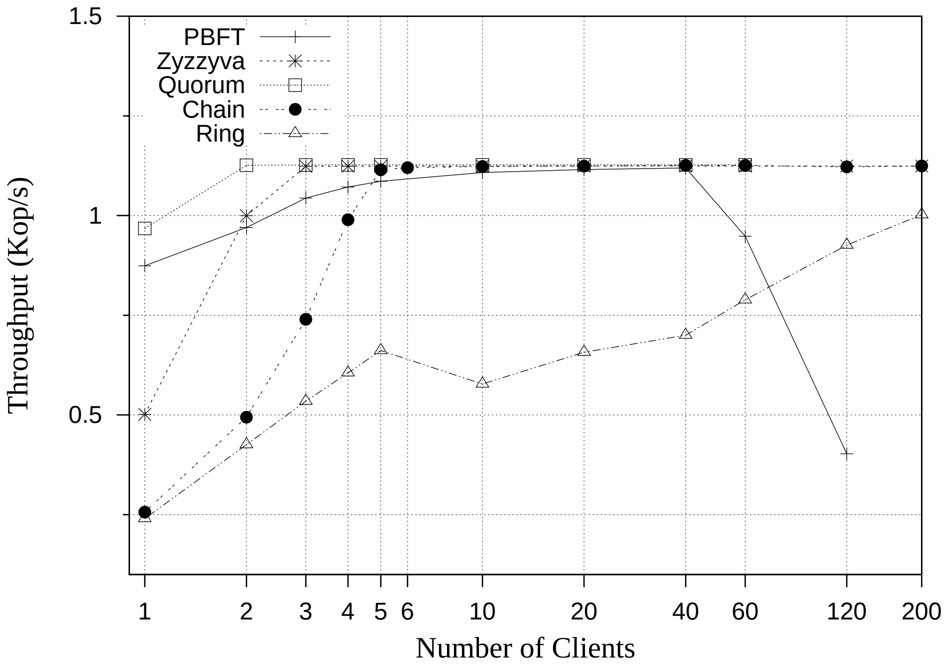


Figure 3.14: Throughput of concurrent clients where T_e is high, using the 1/1 microbenchmark.

$$t_q = \frac{\delta/\delta}{\mu/\delta|\mu - \delta|} = \frac{1}{\mu|\mu/\delta - 1|}$$

And then, $t_q \approx \frac{1}{\mu} = T_p$. Since the system is unstable in this case, we added the absolute value to $(\mu - \delta)$ since we allow for arrival rates larger than servicing rate, which is usually not allowed in stable systems for a long duration of time. In order to avoid infinite queue sizes, system designers usually define a finite capacity for the queues. Supposing that the maximum capacity of the queue is N, then under high contention the queue gets full, and thus, the waiting time of the last request in the queue (i.e., the tail) will be $N \times t_q$. Therefore, equation 3.3 becomes:

$$T_{total} \approx T_d + (m + 1 + N)T_p \tag{3.5}$$

Under these conditions, the system becomes saturated, and thus the whole route of the requests becomes a filled tube (this is hard to happen in WANs). Then, the throughput of the system depends on the flux of the narrowest part of the tube, i.e., the bottleneck. In our system, two possible bottlenecks might be reached: CPU bottleneck, and network bottleneck.

CPU is the bottleneck. This occurs when T_e is high. In the previous sections, we explained that protocols with a small number of authentications achieve higher peak throughput. Nevertheless, if T_e is large, the difference in performance among protocols becomes negligible. In fact, since $T_p = T_t + T_e + T_a + T_h$ and T_e is large, then T_e dominates the other factors. On the other hand, T_p is multiplied by $m + 1 + N$ according to equation 3.5, then the impact of T_p dominates T_d. Consequently, T_e forms a CPU bottleneck on the most overloaded replica (usually the primary one). In Figures 3.13 and 3.14 above, we have shown that the curves keep increasing until reaching the same plateau as soon as the CPU becomes the bottleneck. The same result can be noticed in Figure 3.15 too. The histogram shows that the throughput values of the compared protocols are very close when the CPU bottleneck is reached. In the same figure, Ring shows some shortage in the peak throughput, this is justified by the long messaging pattern (see Figure 3.5) of Ring being designed to work under high contention. However, other experiments show that Ring reaches the same peak throughput of the other protocols with a higher number of clients (almost 400 clients). Moreover, Figure 3.15 conveys the fact that the impact of message payload size becomes negligible too. This occurs since T_e dominates other processing factors that are usually affected by the message size; hence, the peak throughput of both payload sizes 0KB and 1KB are almost equivalent.

Network is the bottleneck. It can happen that the transmission speed of the network is slower than the processing speed of requests (e.g., in Fast Ethernet). In this case, the NIC interface of the bottleneck replica (usually the primary) gets jammed with waiting requests due to slow transmission, and thus the protocols reach their peak throughput. The analysis follows the same reasoning as the previous section; however, T_t dominates the other factors in this case. Some protocols like Ring circumvents this issue by allowing all the replicas to receive requests, and thus, establishing load balancing among replicas. Ring achieves up to 25% higher throughput than other protocols in some cases. Figure 3.16 presents the logarithmic scale of the throughput of some state of the art protocols using 4KB paylod size [2]. The figure shows that other protocols outperform Ring with up to 60 clients. However, as soon as the network gets jammed with requests, it becomes the bottleneck, and hence Ring dominates the other protocols by more than 25%. We experimented Ring with up to 420 clients and noticed that its throughput remains increasing while the other protocols reach their peak throughput early.

[2] We used this payload size to overload the network interfaces.

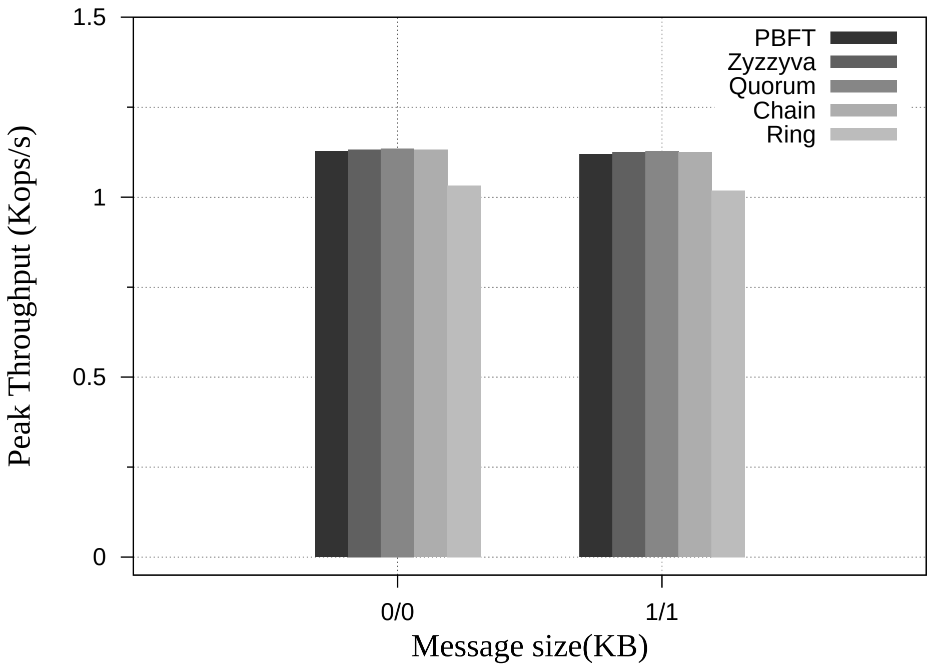


Figure 3.15: Peak throughput when the CPU is the bottleneck.

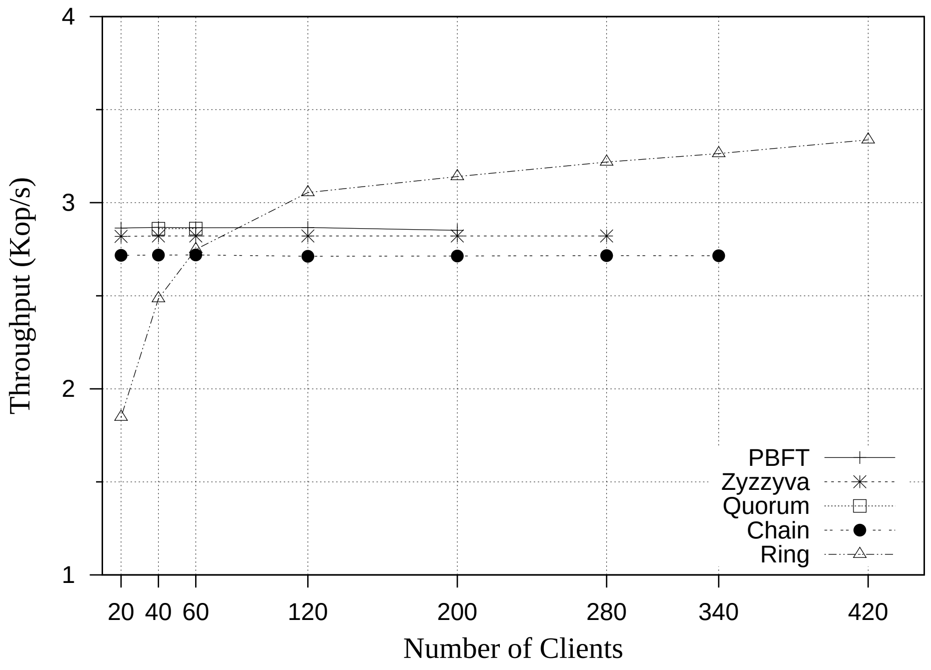


Figure 3.16: Peak throughput when the network is the bottleneck using the 4/4 microbenchmark.

3.4.4 Other Measures

BFT protocols differ in other measures too. For instance, quorum-based protocols like Quorum and Q/U could not tolerate a high number of clients as primary-based protocols. This can be explained by the fact that the latter have a greater tendency to order requests under contention while quorum-based protocols suffer from extensive recovery when different versions (on replicas) of the same object skew.

On the contrary, quorum-based protocols achieve higher fault scalability than primary-based protocols, i.e., the throughput of the system keeps increasing as f (the number of assumed faulty replicas) increases. In fact, the motivation behind Q/U was to improve fault scalability [2]. By increasing f, the system requires more replicas (i.e., at least $3f+1$), and consequently inter-replica communication increases. Thus, increasing f has a greater impact on other protocols since Quorum and Q/U do not rely on inter-replica communication.

In addition, BFT protocols differ in performance in the presence of faults. For instance, though Zyzzyva outperforms PBFT in failure-free cases, PBFT performs better than Zyzzyva under failures by 15% [32]. This is caused by the high number of message exchanges required during the recovery phase of Zyzzyva in order to maintain consistency under failures [32]. In addition, Q/U uses a very expensive recovery case, whereas, *abortable* protocols use a PBFT-like recovery phase.

Further measures that explore the discrepancy among BFT protocols also exist, however, we believe that the points we covered in this chapter are enough to explain the motivation of this thesis.

3.5 Conclusion

The goal of this chapter was to to make a strong belief that: one-size-fits-all BFT protocols are hard to achieve. We started by showing some characteristics of BFT protocols that make them different from each other; and thus choosing one protocols could result in sacrificing important characteristics that other protocols can provide. Then we introduced a theoretical analysis considering basic BFT properties like: number of replicas required, number of authentications needed, messaging delays, etc. This analysis explored the discrepancy in performance among BFT protocols and conveyed that no one protocols dominates the others in all measures. In addition, we conducted an experimental analysis accompanied with a justification using queuing theory methods. In this analysis, we highlighted some cases where BFT protocols clearly differ in performance, and other cases where the difference in performance is negligible.

We provided many experimental figures to support our analysis and convey that the theoretical analysis complies with the practical results. Before this, we introduced the experimental settings and software used in our experiments (i.e., benchmarks, Redis, and OpenLDAP). We addressed the integration of OpenLDAP with BFT in more details being one of our contributions.

In brief, this chapter explained the fact that it is hard to find a single BFT protocol that is convenient to all situations; consequently, other solutions are intuitively demanded. The rest of this thesis describes our solution.

Chapter 4

BFT Quality of Service

This chapter presents a solution to the problems introduced in the previous chapter, and thus helps BFT users to choose their 'preferred' BFT protocol. For this sake, we introduce a BFT selection model and a selection algorithm to choose the most convenient protocol according to user preferences. The selection algorithm uses some matrix formulas to make selection easy and automatic. The chapter describes three different selection modes: static, dynamic, and heuristic. These modes differ according to the underlying conditions of the system. The static mode addresses the cases where a single protocol is needed; the dynamic mode assumes the system conditions are quite fluctuating; the heuristic mode is similar to the dynamic mode but it uses additional heuristics to improve performance. For the dynamic mode, we discuss when it is worthy to change a running protocol. In addition, the chapter provides many concrete cases to evaluate our approach and to describe how it can be used efficiently.

4.1 Introduction

Many BFT protocols have been introduced to fill the gaps of their predecessors as shown in Chapter 2; however, each new protocol had to sacrifice some of the properties of the others too. Consequently, the discussion in Chapter 3 shows that one-fits-all protocols are hard to achieve in Byzantine fault tolerance (BFT). Furthermore, the wide range of characteristics of existing BFT protocols makes it confusing for users to choose the protocol that fits their demands the best. Thus, the following questions arise: What is the 'preferred' BFT protocol? And before, what is meant by the word 'preferred'? A 'preferred' protocol is always the 'preferred'? If not, is it worthy to move to another 'preferred' protocol?

We believe that selection mechanisms are necessary to help in decision making towards the 'preferred' BFT protocol to adopt. Inspiring from software engineering aspects that a successful product is the one which can match customer demands and his satisfaction; we present a solution that chooses the preferred protocol considering user preferences. Hence, the preferred protocol is the preferred one in the eyes of a certain client (i.e., a BFT user), and not necessarily the preferred for other clients. Figure 4.1 depicts a matching problems between five protocols of different preferences with two preferences of different users. The arrows in the figure demonstrate how different protocols can be chosen as 'preferred' for different users. Thus, in this chapter, we define a BFT selection model that matches the properties of BFT protocols across the demands of the user. In addition, we present

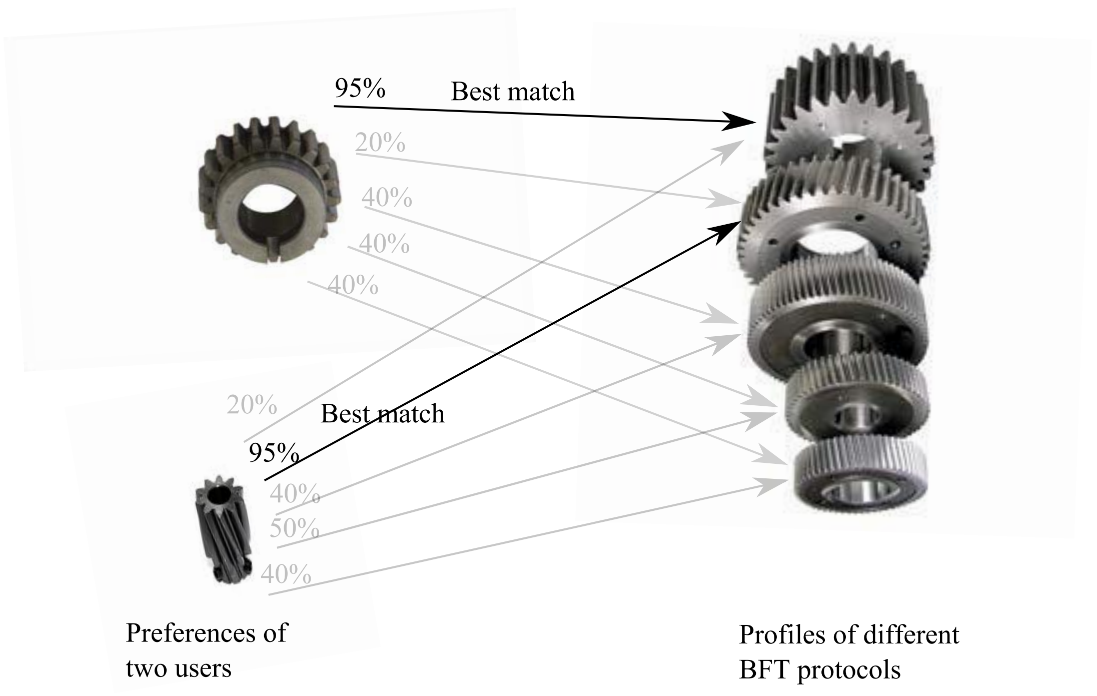


Figure 4.1: Schematic matching process between two user preferences and five profiles of different BFT protocols.

a selection algorithm that can operate in three modes: static, dynamic, and heuristic. The static mode can be used in stable services, i.e., where choosing a protocol is needed only once; whereas, the dynamic mode handles the situation where different alternating protocols are needed in a short period of time due to the frequent changes in the system state. Furthermore, we show how the former can be useful in IaaS, PaaS, and SaaS Clouds, where Cloud vendors present BFT services along with the original service, and users define their BFT requirements in the SLA contract. The dynamic mode, however, is useful in abortable BFT systems, where a real-time dynamic switching is needed to switch from one protocol to another according to the changes in the underlying system state and conditions. The heuristic mode is an advanced dynamic mode that can adjust the user preferences according to the changes in the system state. We recall some related work in the next section; and then, we introduce the BFT QoS selection model and algorithm in the following sections.

4.2 Related Work

In this section we recall some related work that can help understanding our ideas. The related concepts to our approach are divided into three parts: (1) Switching between protocols, (2) QoS selection, and (3) cloud computing.

4.2.1 Switching Between Protocols

Research community is always introducing new protocols in order to enhance robustness and performance of existing BFT protocols, however, the idea of using multiple protocols at once did not get a great attention. Guerraoui et al. introduced the abortability approach in [27] and [26] to allow switching from one protocol to another as the underlying conditions change. However, no clear switching mechanism was introduced to dispatch these protocols, instead, a trivial method was used by switching to a specific protocol according to a predefined order and only when problems exist. This chapter introduces a BFT QoS model that provides method to select the most convenient protocol according to the user. The approach considers both static switching and dynamic switching.

4.2.2 QoS Selection

QoS selection models and algorithms gained a big interest from research community with the increasing popularity of Web Services. The work of Wang et al. [53] represents the base of QoS (Quality of Service) selection models and algorithms. In this work, the author introduces a QoS selection model and algorithm for the WSMO semantic model of Web Services. In general, the selection mechanism tries to match the requirements of the user with the evaluation score of the proposed services. The existing services are evaluated as in Equation 4.1, where q_{ij} and w_i represent the value and the weight of a property j in any service i, respectively.

$$M = \sum_{i=1}^{m}(q_{ij} \times w_j) \tag{4.1}$$

In this chapter, we provide a BFT selection model that inspires many ideas from these existing concepts. In addition, we introduce a new selection algorithm that has some common features to existing algorithms of Web Services (e.g., [53] and [57]). However, our algorithm differs in other aspects that we adjust according to the demands of BFT, in order to improve static and dynamic switching at once and automate the selection process. On the other hand, we provide some mathematical formulas that can be used efficiently to conduct selections. In fact, the early versions of our approach were different than the traditional QoS models, however, we updated it later for two reasons: (1) to stay within the standard notations of QoS being very well-known and mature, and (2) to conserve the WSMO Web Services structure that allows introducing new services to cloud computing, e.g., like BFT.

4.2.3 Cloud Computing

Clouds are becoming everywhere nowadays; they present a quick and cheap (charges are payed in hourly basis) solution to clients (e.g., small and moderate size enterprises). Cloud computing applies virtualization techniques on one or more machines to create other, probably smaller, virtual machines with predefined characteristics like CPU, storage, RAM, etc. It provides three types of services: IaaS (Infrastructure as a Service), PaaS (Platform as a Service), and SaaS (Application as a Service). Cloud vendors (e.g., Amazon [7]) allows clients to use their services through signing an SLA (Service Level Agreement) contract. Therefore, cloud vendors can rent additional features along with the required service like Byzantine Fault Tolerance for instance. Thus, our approach encourages introducing another kind of services (like BFT) by cloud vendors in a similar way to Web Services. In this paper, we present a selection mechanism to help clients decide what are the BFT techniques that can fit their services the best.

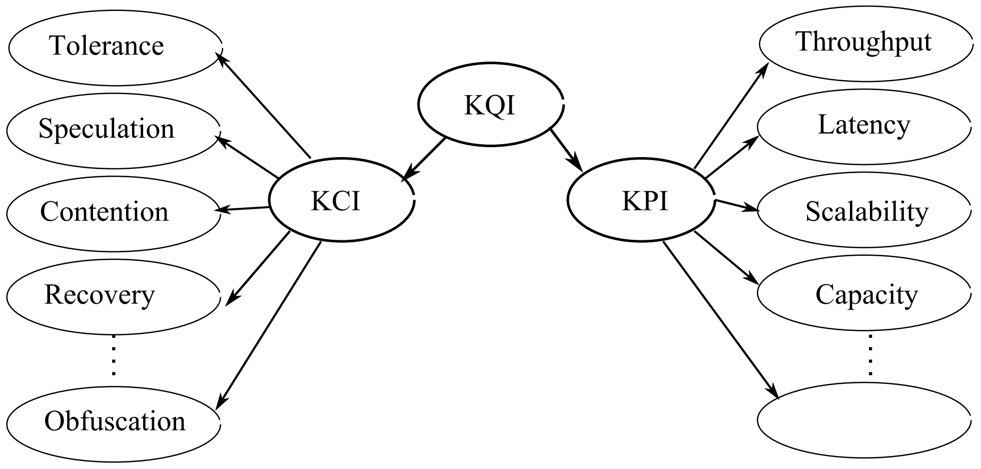


Figure 4.2: Example on the vocabulary of Key Quality Indicators (KQI) class including both Key Characteristic Indicators (KCI) and Key Performance Indicators (KPI).

Class QoS **sub-Class** nonFunctionalProperties
  hasMetricName **type** *string*
  hasValue **type** *value*
  hasMetricValue **type** *value*
  hasMeasurementUnit **type** *Unit*
  hasValueDefinition **type** *logicalExpression*
  isDynamic **type** *boolean*
  isOptional **type** *boolean*
  hasTendency **type** *small, large, given*
  isGroup **type** *boolean*
  hasWeight **type** *string*

Class KQI **sub-Class** QoS
  isHeuristic **type** *boolean*

Table 4.1: The KQI and QoS WSMO classes.

4.3 BFT QoS Notations and Terms

Before diving into the presentation of the selection model, we introduce some terms and notations that will be used in this model. First, we call any property that specifies the quality of a BFT protocol by Key Quality Indicator (KQI). A KQI is composed of two types of indicators: Key Characteristic indicators (KCI), and Key Performance Indicators (KPI). Figure 4.2 represents an example on the KQIs vocabulary. KCIs are those discrete (boolean) properties of a protocol that indicate its behavior, properties, requirements, e.g., number of replicas needed, tolerates malicious clients, obfuscated, etc. The value of this type of indicators can strictly decide whether an evaluated service could be selected or not. The KPIs are the properties that evaluate the performance of a certain protocol like throughput, latency, scalability, etc. These values are usually continuous (real) values. KPIs are usually used to recommend a service over another but, in general, it could not rule out a service.

To make the notations more generic and to comply with the Web Services standards, we define the KQI WSMO class, an inheritance of the QoS class defined in [53]; the two classes are depicted in Table 4.1. In Figure 4.3, we present the interpretation of the properties of KQI and QoS WSMO classes.

The definition and the discussion of concrete measurements of KCIs is out of the scope of this thesis.

hasMetricName. this defines the name of the property, e.g, throughput.

hasValue. types of this parameter may be linguistic, numeric (int, float, long), boolean (0/1, True/False) or other. There will be different forms of preprocessing according to the different value types.

hasMetricValue. this property defines a metric's values which are either real ones or a string such as 'calculate'. If MetricValue = 'calculate', then this attribute should refer to its valueDefinition for a dynamic value calculation.

hasMeasurementUnit. this property specifies the concrete unit of every quality metric, with possible types such as Unit = $, millisecond, percentage, kpbs, times, ...

hasValueDefinition. this either a logical expression or the string 'NULL'. If hasValueDefinition = 'NULL', then this value definition cannot explicitly extracted from the context of service description, but must dynamically be invoked from its service provider. In this case this quality attribute must be dynamic, that is isDynamic = True.

isDynamic. through this property, the nature of a quality is defined as static or dynamic. For a static quality, its values are given in priori, and can be directly used during the selection process. If isDynamic = True, this quality metric must be dynamically invoked and obtained from its service provider, or/and its values must be calculated at run-time.

isOptional. this parameter tells whether the current property is mandatory for the service or it is optional. If isOptional = 'False', then the absence of the current property harms the service, consequently, the current service is out of choice.

hasTendency. this is an object property representing the expected tendency of the value from the user's perspective. For example, the latency of a service is expected to be as low as possible, so that its hasTendency = 'low/small'. On the contrary, the throughput of a service should be as high as possible, i.e., hasTendency ='high/large'. When hasTendency ='given', the user expects the value of this quality to be as close to the given value as possible.

isGroup. this indicates if this quality attribute is defined by a group of other qualities or not. For example, suppose that security is composed of nonRepudiatior, DataEncryption, Authorization, Authentication, Auditability, and Confidentiality. Hence, isGroup = True means that in the preprocessing stage, the group value must be calculated first.

hasWeight. this is a value denotes the impact of the property with respect to the others, especially when synthetically measuring several metrics. The weight value either ranges in [0, 10] or 'NULL'. Different end users have different weight values for their service requirements. If hasMetricValue = 'calculate' holds, its hasValueDefinition property must be checked to determine how to calculate it. If hasValueDefinition ='NULL' and isDynamic = 1, then the invocation function is to inquire the real-time value. If isDynamic = 0, its corresponding hasMetricValue is an existing value. If isAdaptable= 'True', this value will change according to some heuristics due to system state changes.

isHeuristic. this value designates whether this property is adaptable to the changes of the system state using predefined heuristics. If isHeuristic= 'True', then the hasWeight of this parameters can be changed according to the external heuristics of the system. This parameter is designed to systems that impose different behavior due to frequent different conditions.

Figure 4.3: Interpretation of the parameters of KQI and QoS WSMO classes.

4.4 BFT Selection Model

Consider a service provider (e.g., a cloud) that provides n different BFT protocols for users. We define the set of protocols $\psi = \{p_i;\ where\ 1 \leq i \leq n\}$; where p_i is one of the BFT protocols. On the other hand, consider a selection model represented by:

$$\Sigma = \{PROTOCOL, USER, MODE\} \tag{4.2}$$

where PROTOCOL represents the profile of a BFT protocol, USER represents the preferences of the user, and MODE represents the selection mode of the system. Selection occurs through matching the PROTOCOL profile with the USER preferences according to the mapping: $f : \Sigma \longmapsto \psi$; this yields the 'preferred' protocol among all competing protocols. Here we introduce the definition of 'preferred':

Definition 4.1. *A protocol p_i with profile $PROTOCOL_i$ is called the 'preferred' protocol among a set of available protocols ψ with respect to a specific user with preferences $USER_j$ if and only if according to an evaluation function $e : \Sigma \longmapsto \Re$, e($PROTOCOL_i$, $USER_j$,ϕ) is maximal.*

The interpretation of PROTOCOL, USER, and MODE is as follows:

PROTOCOL. Each protocol is described in a profile of KQIs and default weights: $PROTOCOL = \{A_P, A_U, B_P, B_V\}$, where A_P is a vector of a KCIs: $A_P = (\alpha_1, \alpha_2, ..., \alpha_a)$, and A_U represents the vector of the corresponding default weights of these KCIs: $A_U = (u_1, u_2, ..., u_a)$. B_P, however, is a vector of b KPIs: $B_P = (\beta_1, \beta_2, ..., \beta_b)$, and B_V represents the vector of the corresponding default weights of these KPIs: $B_V = (v_1, v_2, ..., v_b)$.

USER. Each user defines his preferences in $USER = \{U, V, M\}$, where U (resp., V) is a vector of user defined weights corresponding to the KCIs (resp., KPIs) of the PROTOCOL's preference A_P (resp., B_P). M defines the mode required by the user, i.e, either DYNAMIC, STATIC, or HEURISTIC.

MODE. The selection can occur in three different modes: STATIC, DYNAMIC, or HEURISTIC. In the former, the selection occurs only once, i.e., at the time the user requires a service; afterwards, the user does not change his selection (i.e., the used protocol) until the system is halted. Thus, he provokes a new selection. On the other hand, the DYNAMIC mode tells the system to react dynamically to changes of the system state. This mode allows the system to adapt to the upcoming conditions and thus the user will be using multiple protocols at once. The HEURISTIC mode is similar to the DYNAMIC mode but, in addition, it uses some heuristics to adjust the preferences of the user, especially V. The heuristics are represented in a vector W similar to U and V, but its values are modified as the system state changes according to some predefined heuristics. The default mode is STATIC.

In the STATIC mode, A_P and B_P are predefined. They are calculated before selection occurs, and even before the advertisement of the protocol is done. However, in the DYNAMIC and HEURISTIC modes, the values of B_P change dynamically while the system is running. In this thesis, for instance, we use predictions to assess these values in real-time. We define the system state by $S = \{s_i = (f_1, f_2, ..., f_j, ..., f_m)\}$ where f_j represents the j^{th}

impact factor of the system state and m is the number of considered impact factors by the system. When these impact factors change, the system state changes; consequently, a new dynamic evaluation and selection is needed to move, probably, to a new protocol that performs better than the current one under the new conditions. We explain this mechanism further in later sections.

4.5 Selection Mechanism

The selection mechanism of the most convenient protocol according to the user preferences, i.e., the preferred protocol, is achieved through computing the evaluation score E of the competing protocols, and then electing the protocol that corresponds to the maximum score. As mentioned in the previous section, the selection process is represented by the mapping: $f : \Sigma \longmapsto \psi$; Equation 4.3 is an example of such mapping.

Selection (and evaluation) occurs only once in the STATIC mode, i.e., at deployment time. Otherwise, the deployed service should be stopped and restarted again, probably, using another selected protocol if the user preferences changed. If the mode of the system is DYNAMIC or HEURISTIC, then the evaluation process of protocols can take place at any instant. The system chooses the protocol that has the highest evaluation score E among all protocols to launch it in the next phase. Formally speaking, for any state s, and protocol $p_i \in \psi$ that has an evaluation score $E_{i,s}$; a protocol p_{next} is chosen according to the function:

$$p_{next} = p_i, \ s.t. \ E_{i,s} = \underset{1 \leq j \leq n}{max} E_{j,s}. \tag{4.3}$$

The evaluation scores are computed in real-time. When the system state changes, the evaluation of the protocols E also changes. Then, the protocol p_{next} is launched in the next phase only if it is worthy to switch (see later).

4.5.1 Definitions

To automate the evaluation process, the mechanism depends on mathematical formulations using matrices. Before whelming into the evaluation formulas, and to make them easier to understand, we start by recalling some mathematical concepts and defining new ones.

Definition 4.2. *Consider two matrices of the same dimension $A, B \in \Re^{m \times n}$ with entries a_{ij}, and b_{ij}, respectively. The Schur product $A \circ B$ (also known as Hadamard product) is a matrix of the same dimensions $C = A \circ B \in \Re^{m \times n}$, where its elements are defined by: $c_{ij} = a_{ij}.b_{ij}$.*

Example 4.1. *Consider the two matrices:*

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}; \ and \ B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{pmatrix}$$

The Schur product $C = A \circ B$ is as follows:

$$A \circ B = \begin{pmatrix} a_{11}.b_{11} & a_{12}.b_{12} \\ a_{21}.b_{21} & a_{22}.b_{22} \end{pmatrix}$$

Definition 4.3. *Consider a matrix $A \in \Re^{m\times n}$ with entries a_{ij} and a scalar $\lambda \in \Re$. The Scalar product $B = \lambda.A$ is a matrix of the same dimension as A, where its elements are defined by: $b_{ij} = \lambda.a_{ij}$.*

Example 4.2. *Consider a scalar $\lambda \in \Re$, and the matrix:*

$$A = \begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix}$$

The Scalar product $B = \lambda.A$ is as follows:

$$\lambda.A = \begin{pmatrix} \lambda.a_{11} & \lambda.a_{12} \\ \lambda.a_{21} & \lambda.a_{22} \end{pmatrix}$$

Definition 4.4. *Consider two matrices $A \in \Re^{n\times l}, B \in \Re^{l\times m}$ with entries a_{ij}, and b_{ij}, respectively. The product AB is a matrix $C = AB \in \Re^{n\times m}$, where its elements are defined by: $c_{ij} = \sum_{k=1}^{m} a_{ik}.b_{kj}$.*

Example 4.3. *Consider the two matrices:*

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{1,3} \\ a_{21} & a_{22} & a_{2,3} \end{pmatrix}; \text{ and } B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix}$$

The product $C = AB$ is as follows:

$$AB = \begin{pmatrix} a_{11}.b_{11} + a_{12}.b_{21} + a_{13}.b_{31} & a_{11}.b_{12} + a_{12}.b_{22} + a_{13}.b_{32} \\ a_{21}.b_{11} + a_{22}.b_{21} + a_{23}.b_{31} & a_{21}.b_{12} + a_{22}.b_{22} + a_{23}.b_{32} \end{pmatrix}$$

In addition, to make computations easier, we define a new operator, i.e., the OR product $\dot{\vee}$.

Definition 4.5. *Consider two boolean matrices $A \in \{0,1\}^{n\times l}, B \in \{0,1\}^{l\times m}$ with entries a_{ij}, and b_{ij}, respectively. The OR product $A\dot{\vee}B$ is a matrix $C = A\dot{\vee}B \in \mathbb{N}^{n\times m}$, where its elements are defined by: $c_{ij} = \sum_{k=1}^{m} a_{ik} \vee b_{kj}$. The operator $\vee$ is the logical OR operator.*

Example 4.4. *Consider the two boolean matrices:*

$$A = \begin{pmatrix} a_{11} & a_{12} & a_{1,3} \\ a_{21} & a_{22} & a_{2,3} \end{pmatrix}; \text{ and } B = \begin{pmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \\ b_{31} & b_{32} \end{pmatrix}$$

The product $C = A\dot{\vee}B$ is as follows:

$$A\dot{\vee}B = \begin{pmatrix} a_{11} \vee b_{11} + a_{12} \vee b_{21} + a_{13} \vee b_{31} & a_{11} \vee b_{12} + a_{12} \vee b_{22} + a_{13} \vee b_{32} \\ a_{21} \vee b_{11} + a_{22} \vee b_{21} + a_{23} \vee b_{31} & a_{21} \vee b_{12} + a_{22} \vee b_{22} + a_{23} \vee b_{32} \end{pmatrix}$$

4.5.2 Evaluation of Protocols

As mentioned above, the selection of the preferred protocol is represented by the mapping: $f : \Sigma \longmapsto \psi$; where, $f = p_{next}$ in Equation 4.3 is an example of such mapping. The evaluation score E is calculated according to the formulas introduced in Equation 4.4. In the following, we present the interpretation and discussion on these formulas. To make the idea easier to understand, we conduct some concrete examples along with the discussion.

$$\begin{cases} E = D \circ C \\ where\ D = \left\lfloor \frac{1}{a}.A \dot{\vee} \ (e_n - U) \right\rfloor \\ and\ C = B^{\pm}.(V \circ W). \end{cases} \tag{4.4}$$

4.5.2.1 The Evaluation Matrix *E*

The evaluation matrix E is the Schur product (see Definition 4.2 and Example 4.1) of discrete matrix D and the continuous matrix C. D represents the part of the evaluation that deals with the KCIs of the profiles of the protocols; whereas, C represents the evaluation part that deals with the KPIs. E is calculated after computing the values of D and C. If the mode of the system is DYNAMIC or HEURISTIC, then E may change in real-time as C changes.

4.5.2.2 The Discrete Matrix *D*.

In this section, we discuss the discrete matrix: $D = \left\lfloor \frac{1}{a}.A \dot{\vee} \ (e_n - U) \right\rfloor$. This matrix matches the user preferences against the profiles of different protocols. a represents the dimension of KCIs vector (see Section 4.3). The operator $\lfloor\ \rfloor$ is the absolute value operator (it is sometimes indicated by $[\ ]$ too). The operator $\dot{\vee}$ was defined in Definition 4.5. The matrices A, U, and e_n are explained next.

The profile matrix *A*. This matrix represents the profiles of the protocols; it takes the form:

$$A = \begin{pmatrix} \alpha_{1,1} & \alpha_{1,2} & \alpha_{1,a} \\ \alpha_{2,1} & \alpha_{2,2} & \alpha_{2,a} \\ \vdots & \vdots & \vdots \\ \alpha_{n,1} & \alpha_{n,2} & \alpha_{n,a} \end{pmatrix}$$

The dimension of the matrix A is $n \times a$; where n is the number of available protocols (i.e., the cardinality of ψ) and a is the number of KCIs considered in the evaluation. Each row of the matrix represents a KCI vector profile of a protocol. For instance, consider the KCIs of state-of-the-art protocols depicted in Table 4.2; the boolean representation of these values are depicted in Table 4.3. Obtaining the latter table is trivial. If a protocol has a certain property, i.e., has the value 'Yes', it takes the value 1; otherwise, it lacks that property and thus takes the value 0. Note that, the values in the table are not necessarily 100% accurate but they are acceptable to conduct our examples. Deciding the values of KCIs is out of the scope of this thesis.

Example 4.5. *According to the boolean representation of KCIs in Table 4.3, the matrix A can be defined as follows:*

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}$$

Notice that, each column (i.e., protocol) in Table 4.3 represents a row in the matrix.

	PBFT	Zyzzyva	Q/U	HQ	Quorum	Ring	OBFT	Chain
Non-speculative	**Yes**	No	No	No	No	No	No	No
Byzantine clients	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	**Yes**	No	**Yes**
Tolerates contention	**Yes**	**Yes**	No	**Yes**	No	**Yes**	**Yes**	**Yes**
No IP Multicast	**Yes**	**Yes**	No	**Yes**	No	**Yes**	**Yes**	**Yes**
Self-Recovery	**Yes**	**Yes**	**Yes**	**Yes**	No	**Yes**	**Yes**	No
Obfuscated	No	No	**Yes**	No	**Yes**	No	**Yes**	No
Requires $\leq 3f+1$	**Yes**	**Yes**	No	**Yes**	**Yes**	**Yes**	No	**Yes**

Table 4.2: Some discrete properties (KCIs) of different state-of-the-art BFT protocols.

	PBFT	Zyzzyva	Q/U	HQ	Quorum	Ring	OBFT	Chain
Non-speculative	**1**	0	0	0	0	0	0	0
Byzantine clients	**1**	**1**	**1**	**1**	**1**	**1**	0	**1**
Tolerates contention	**1**	**1**	0	**1**	0	**1**	**1**	**1**
No IP Multicast	**1**	**1**	0	**1**	0	**1**	**1**	**1**
Self-Recovery	**1**	**1**	**1**	**1**	0	**1**	**1**	0
Obfuscated	0	0	**1**	0	**1**	0	**1**	0
Requires $\leq 3f+1$	**1**	**1**	0	**1**	**1**	**1**	0	**1**

Table 4.3: Boolean representation of KCIs of different state-of-the-art BFT protocols.

The user preferences matrix U. This is a vector matrix that represents the requirements of the user. According to this vector, the protocols that satisfy all these requirements will be considered for selection (i.e., will continue the competition). On the contrary, the protocol that lacks a single property among those demanded by the user will be out of selection. It can happen that the user does not provide his requirements because he does not have enough experience to do so, and thus he keeps the decision for the specialists of the service provider (e.g., cloud vendors). This is possible through choosing the default vector A_U provided by the system (see Section 6.4). U is a 1-column matrix (of dimension $a \times 1$) that takes the form:

$$U = \begin{pmatrix} u_{1,1} \\ u_{2,1} \\ \vdots \\ u_{a,1} \end{pmatrix}$$

Example 4.6. *The following matrix represents the default user KCI preferences vector. When the user does not define his requirements, this vector is used automatically.*

$$U = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

The values of this vector correspond to the KCIs (Non-speculative, Byzantine clients, Tolerates, Contention, No IP Multicast, Self-Recovery, Obfuscated, Requires $\leq 3f+1$). This vector tells that the user needs a protocol that is not speculative and tolerates Byzantine clients, and that he does not care for the other properties (in our case, PBFT is the only protocol that satisfies these condition, thus PBFT is default).

Note that, if the user chooses the vector $U = e_a$, then any discrete matrix has no effect on the selection, and the protocols are only selected regarding the continuous matrix. On the contrary, if $U = 0_a$, then no protocol will be selected unless it satisfies every property. In our case, non of the protocols satisfy this condition, and thus none will be selected.

The matrix $\boldsymbol{e_n}$. This is a column matrix of dimension $a \times 1$. Each entry of this matrix takes the value 1:

$$e_a = \begin{pmatrix} 1 \\ 1 \\ \vdots \\ 1 \end{pmatrix}$$

We use this matrix is to invert the values of the matrix U to $-U$.

Calculating $\boldsymbol{D}$. After defining the matrices in the formula $D = \left\lfloor \frac{1}{a}.A \dot{\vee} (e_a - U) \right\rfloor$, the computation D becomes trivial. To explain the process, we conduct the following example.

Example 4.7. *In this example, we compute the matrix D given the two matrices:*

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix}; and\ U = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

The computation is as follows:

$$\begin{aligned}
D &= \left\lfloor \tfrac{1}{a}.A \ \dot{\vee} \ (e_a - U) \right\rfloor \\
&= \left\lfloor \tfrac{1}{7}. \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \ \dot{\vee} \ \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 1 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \right\rfloor & (step\ 0) \\
&= \left\lfloor \tfrac{1}{7}. \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 & 1 & 0 & 1 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 \end{pmatrix} \ \dot{\vee} \ \begin{pmatrix} 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} \right\rfloor & (step\ 1) \\
&= \left\lfloor \tfrac{1}{7}. \begin{pmatrix} 7 \\ 6 \\ 6 \\ 6 \\ 6 \\ 6 \\ 5 \\ 6 \end{pmatrix} \right\rfloor & (step\ 2) \\
&= \left\lfloor \begin{pmatrix} 1 \\ 6/7 \\ 6/7 \\ 6/7 \\ 6/7 \\ 6/7 \\ 5/7 \\ 6/7 \end{pmatrix} \right\rfloor = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} & (step\ 3)
\end{aligned}$$

Step 1 computes the difference ($e_a - U$). This inverts the values of U (i.e., a boolean x becomes $\neg x$). In step 2, the OR product is calculated (recall the OR product in Definition 4.5 and Example 4.4). Notice that, each KCI in the protocol profiles (i.e., each row) is multiplied (OR multiplication) by the corresponding items of the user's preferences. The resulted entries represent the sum of the properties that each protocol supports. In step 3, the absolute value of these sums are calculated, and the final matrix D is achieved.

Through analyzing Example 4.7, important notes can be drawn. First, notice that the computation is automated using these formulas. Imagine the time and effort needed to achieve the result of the matrix D by hand, especially, when the requirements vector U is more complicated. In addition, the selection depends, strictly, on the user preferences. In our example, PBFT was the only selected protocol (which is the default choice). However, protocols of important characteristics can be evicted out of competition according to the user preferences. For instance, Zyzzyva was discarded in Example 4.7 since it is a speculative protocol (see the 2^{nd} entry of U).

Notice that the selection process using the discrete matrix D alone is correct; however, it does not take into consideration the performance of the protocols, i.e., the KPIs. The continuous matrix C is derived then to achieve this goal.

4.5.2.3 The Continuous Matrix C.

The continuous matrix is used to update the discrete selection by considering the KPIs of the protocols seeking better performance. The continuous matrix C is defined in the formula: $C = B^{\pm}.(V \circ W)$. The matrices B, V, and W are discussed in the following.

The matrix $B^{\pm}$. This matrix represents the KPI profiles of each protocol. Each profile is presented in one row. This matrix is a normalized version of another matrix B. B and $B^{\pm}$ have the same dimension $n \times b$ where n is the number of protocols and b is the number of KPIs considered, i.e., the dimension of the KPI vector defined in Section 4.3. The entries of the matrix $B^{\pm}$ are denoted by $\beta^{\pm}$ and are calculated from the entries of B that are denoted by β. Thus, the matrices B and $B^{\pm}$ are of the form:

$$B^{\pm} = \begin{pmatrix} \beta^{\pm}_{11} & \beta^{\pm}_{12} & \beta^{\pm}_{1b} \\ \beta^{\pm}_{21} & \beta^{\pm}_{22} & \beta^{\pm}_{2b} \\ \vdots & \vdots & \vdots \\ \beta^{\pm}_{n1} & \beta^{\pm}_{n2} & \beta^{\pm}_{nb} \end{pmatrix}; B = \begin{pmatrix} \beta_{11} & \beta_{12} & \beta_{1b} \\ \beta_{21} & \beta_{22} & \beta_{2b} \\ \vdots & \vdots & \vdots \\ \beta_{n1} & \beta_{n2} & \beta_{nb} \end{pmatrix}$$

The notion of $\beta^{\pm}$ KPIs. The $\beta^{\pm}$ values of matrix $B^{\pm}$ can be either β^{+}, or β^{-}, they are normalized values of the entries of B (i.e., they belong to the interval [0,1]). They differ according to properties of the KPI considered. If a KPI has the property hasTendency='high' (see Section 4.3), e.g., throughput, then this KPI is of type β^{+}, since a higher throughput means better evaluation score E. On the contrary, if the KPI has the property hasTendency='low', e.g., latency, then this KPI is of type β^{-}, since a higher latency means a worst evaluation score E.

Suppose the number of β-KPIs is b, then the matrix B can be divided into b column matrices (i.e., vectors): $B_1, B_2, B_i, ...,$ and B_b. Let the maximum (resp., minimum) value of the entries of each vector B_i be max_i (resp., min_i). Then, the entries of the matrix $B^{\pm}$ can be calculated as follows:

$$\begin{cases} \beta_{ij}^{+} = 1 - \dfrac{max_i - \beta_{ij}}{max_i - min_i}; \\ \beta_{ij}^{-} = 1 - \dfrac{\beta_{ij} - min_i}{max_i - min_i}; \\ where\ i \leq b\ and\ j \leq n. \end{cases} \tag{4.5}$$

Table 4.4 conveys an analytic evaluation of the studied protocols. Based on this table, we obtain the KPI values β of Table 4.5 considering three KPIs: Throughput, Latency, and Capacity (i.e., the estimated number of clients that can be tolerated). The values of Table 4.5 are calculated as follows: Throughput=10/C, and Latency=D. We use 10/C (and not 1/C) in throughput to show the effect of normalization in the next sections. As for Capacity, we estimate this value based on the message patterns of the protocols (see Chapter 3, Figure 3.5) and some concrete experiments that we conducted for this sake. Note that the values of Table 4.5 can be unaccurate; however, our experience shows that they give a good approximation that is enough to explain our idea. Filling this table is out of the scope of this thesis, we rather keep it for future work.

Next, we explain the calculation of $B^{\pm}$ in the following example.

	PBFT	Zyzzyva	Q/U	HQ	Quorum	Ring	OBFT	Chain
A	**4**	**4**	6	**4**	**4**	**4**	**4**	**4**
B	**3**	**3**	6	4	4	4	4	4
C	10	5	6	6	**2**	≈8	**2**	3
D	4	3	**2**	4	**2**	≈9	4	5
E	3	2	**1**	3	**1**	2	**1**	**1**

Table 4.4: Analytic evaluation for the state-of-the-art BFT protocols tolerating f faults using MACs for authentication, and assuming preferred optimization (without batching): A represents the number of replicas needed to tolerate f Byzantine replicas; B is similar to A but excluding witness and backup replicas; C represents the number of MAC operations on the bottleneck replica; D is the number of one-way latencies needed for each request; and E represents the number of send/to kernel calls on the bottleneck replica. Bold entries denote protocols with the lowest known cost.

	PBFT	Zyzzyva	Q/U	HQ	Quorum	Ring	OBFT	Chain
Throughput (β^{+})	**1**	2	1.67	1.67	**5**	1.25	**5**	3.33
Latency (β^{-})	4	3	**2**	4	**2**	**9**	4	5
Capacity (β^{+})	6	7	2	5	**1**	**10**	7	8

Table 4.5: Analytical values of β^{+} and β^{-} for the state-of-the-art BFT protocols.

Example 4.8. *First, we write down the matrix B that is derived from Table 4.5. Each column in the table represents a row in the matrix B:*

$$B = \begin{pmatrix} 1 & 4 & 6 \\ 2 & 3 & 7 \\ 1.67 & 2 & 2 \\ 1.67 & 4 & 5 \\ 5 & 2 & 1 \\ 1.25 & 9 & 10 \\ 5 & 4 & 7 \\ 3.33 & 5 & 8 \end{pmatrix}$$

The matrix B can be divided into 3 column matrices B_1, B_2, and B_3. The corresponding maxima and minima of these column matrices are as follows: (max_1=5,min_1=1), (max_2=9, min_2=2), (max_3=10,min_3=1). The calculation of $\beta^{\pm}$ is done according to the formulas in Equation 4.5. For instance, the $\beta^{\pm}$ KPIs of PBFT are as follows:

- *the throughput KPI (β^+) of PBFT is: β^+_{11}=$(1 - \frac{5-1}{5-1})$=0.*
- *the latency KPI (β^-) of PBFT is: β^+_{12}=$(1 - \frac{4-2}{9-2})$=0.71.*
- *the capacity KPI (β^+) of PBFT is: β^+_{13}=$(1 - \frac{10-6}{10-1})$=0.55.*

The $\beta^{\pm}$ values of the other protocols (i.e., other rows) are calculated in a similar way, and finally we obtain the matrix $B^{\pm}$:

$$B^{\pm} = \begin{pmatrix} 0 & 0.71 & 0.55 \\ 0.25 & 0.86 & 0.67 \\ 0.17 & 1 & 0.11 \\ 0.17 & 0.71 & 0.44 \\ 1 & 1 & 0 \\ 0.06 & 0 & 1 \\ 1 & 0.71 & 0.67 \\ 0.58 & 0.57 & 0.78 \end{pmatrix}$$

The matrix *V*. This matrix represents the user preferences (i.e., the KPI weights) used to recommend a protocol. V is a column matrix of dimension $b \times 1$, where b is the dimension of the number of KPIs considered in the evaluation. The entries of this matrix follow two constraints: (1) all entries $\in$ [0,10], and (2) their sum $\sum_{i=1}^{b} v_{i1} = 10$. Notice that this differs from the preferences of the discrete matrix U, since the values of $U \in \{0, 1\}$. The matrix V takes the form:

$$V = \begin{pmatrix} v_{11} \\ v_{21} \\ \vdots \\ v_{b1} \end{pmatrix}$$

Example 4.9. *As an example of V, we suppose that the requirements of the user are default, i.e., a protocol with acceptable throughput, latency, and capacity (i.e., tolerates a high number of clients); thus the matrix becomes as follows:*

$$V = \begin{pmatrix} 3 \\ 3 \\ 4 \end{pmatrix}$$

The matrix *W*. This matrix is a column matrix used in the HEURISTIC mode only. W is important to adjust the user preferences given in V by considering the system state. Thus, it can happen that the user chooses a 'low' weight for some KPI in V; however, according to the heuristics of the system, the weight is modified by the corresponding value in W to 'high'. In some cases, this improves the performance of the system significantly.

W is column matrix with the same dimension as V and its entries follow the same constraints: (1) all entries $\in$ [0,10], and (2) their sum $\sum_{i=1}^{b} v_{i1} = 10$. The matrix takes the form:

$$W = \begin{pmatrix} w_{11} \\ w_{21} \\ \vdots \\ w_{b1} \end{pmatrix}$$

Since we assume that the default mode is STATIC, then the entries of W are equal to 1, i.e., W=e_b, and the matrix has no effect on the evaluation. Instead, we leave the discussion of W to later sections where the HEURISTIC mode is used.

Computing *C*. After explaining the matrices $B^{\pm}$, V, and W, the calculation of the $C = B^{\pm}.(V \circ W)$ becomes easy. We conduct the following example to show the process of computing ***C***. After explaining the matrices $B^{\pm}$, V, and W, the calculation of the $C = B^{\pm}.(V \circ W)$ becomes easy. We conduct the following example to show the process.

Example 4.10. *In this example, we compute the matrix C given the three matrices:*

$$B^{\pm} = \begin{pmatrix} 0 & 0.71 & 0.55 \\ 0.25 & 0.86 & 0.67 \\ 0.17 & 1 & 0.11 \\ 0.17 & 0.71 & 0.44 \\ 1 & 1 & 0 \\ 0.06 & 0 & 1 \\ 1 & 0.71 & 0.67 \\ 0.58 & 0.57 & 0.89 \end{pmatrix} ;\ V = \begin{pmatrix} 3 \\ 3 \\ 4 \end{pmatrix} ; and\ W = e_b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

Recall that we set W=e_b since we assume that the mode is STATIC. The computation is as follows:

$$\begin{aligned}
C &= B^{\pm}.(V \circ W) \\
&= \begin{pmatrix} 0 & 0.71 & 0.55 \\ 0.25 & 0.86 & 0.67 \\ 0.17 & 1 & 0.11 \\ 0.17 & 0.71 & 0.44 \\ 1 & 1 & 0 \\ 0.06 & 0 & 1 \\ 1 & 0.71 & 0.67 \\ 0.58 & 0.57 & 0.89 \end{pmatrix} . \begin{pmatrix} 3 \\ 3 \\ 4 \end{pmatrix} \circ \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} \quad (step\ 0) \\
&= \begin{pmatrix} 0 & 0.71 & 0.55 \\ 0.25 & 0.86 & 0.67 \\ 0.17 & 1 & 0.11 \\ 0.17 & 0.71 & 0.44 \\ 1 & 1 & 0 \\ 0.06 & 0 & 1 \\ 1 & 0.71 & 0.67 \\ 0.58 & 0.57 & 0.89 \end{pmatrix} . \begin{pmatrix} 3 \\ 3 \\ 4 \end{pmatrix} \quad (step\ 1) \\
&= \begin{pmatrix} 4.33 \\ 6.01 \\ 3.95 \\ 4.4 \\ 6 \\ 4.18 \\ 8.25 \\ 6.57 \end{pmatrix} \quad (step\ 2)
\end{aligned}$$

Step 1 is trivial, it the Schur product of V and W; but since W=e_b, then the Schur product gives V. In Step 2, the product of the two matrices: B and V is performed.

The results of Example 4.10 show the evaluation of the performance of the protocols by considering the three KPIs throughput, latency, and capacity together. The result is reasonable as it conveys that OBFT, Zyzzyva, and Chain achieve the best scores according to the current user preferences in V (recall that OBFT does not tolerate Byzantine clients). The final evaluation of E, however, yields a different result. The reason is that the KCI preferences of the user have to be taken into consideration.

4.5.3 The Preferred Protocol

To achieve the preferred protocol, the evaluation scores of matrix E should be computed. E is a vector matrix such that: $E = D \circ C$. It represents the overall evaluation of KCIs and KPIs together. After calculating E, the 'preferred' protocol with respect to the user can be selected using the equation:

$$p_{next} = p_i, \ s.t. \ E_{i,s} = max(E) = \max_{1 \leq j \leq n} E_{j,s}. \tag{4.6}$$

Where n is the number of protocols considered, and s is the current state of the system. The next example shows the calculation of E.

Example 4.11. *In this example, we put things together. To calculate E, we use the matrices already calculated in Examples 4.7 and 4.10:*

$$D = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} ; \textit{and } C = \begin{pmatrix} 4.33 \\ 6.01 \\ 3.95 \\ 4.4 \\ 6 \\ 4.18 \\ 8.25 \\ 6.57 \end{pmatrix}$$

Then, E is calculated using the Schur product as follows:

$$E = D \circ C = \begin{pmatrix} 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} \circ \begin{pmatrix} 4.33 \\ 6.01 \\ 3.95 \\ 4.4 \\ 6 \\ 4.18 \\ 8.25 \\ 6.57 \end{pmatrix} = \begin{pmatrix} 4.33 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix}$$

The preferred protocol can be achieved then by finding p_{next}. For the system state s=Current, $E_{1,Current} = max(E)$=4.33, and then p_{next}=p_1 =PBFT.

Thus, as shown in Example 4.11, the discrete matrix D forces the choice of PBFT (that corresponds to the evaluation score 4.33) though the other protocols achieve a higher score as shown in matrix C. Of course, choosing another KCI preferences vector U will change this leadership. We do not consider other preferences in the STATIC mode, but we do it in the DYNAMIC and the HEURISTIC modes since these modes are more interesting. Finally, the 'preferred' protocol, i.e., PBFT, is chosen through calculating max(E), which is the score that corresponds to the first protocol in E (i.e., the first entry).

4.5.3.1 DYNAMIC Mode

The DYNAMIC mode is designed for systems that exhibit fluctuating states, and thus the system needs to adapt to the new state dynamically. In this mode, evaluations and selection can occur more than once, i.e., during the normal operation of the system. For instance, consider a system that is exposed to high contention during the day more than nights, or during a season more than other seasons. In this case, moving to a protocol that can perform better under these new conditions will be very useful. The motivation behind the DYNAMIC mode can be noticed clearly through observing the values of KPIs in the matrix B. To explain this, we give the following example.

Example 4.12. *In this example, we suppose the user preferences of KCIs are as follows:*

$$U = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}$$

U imposes no constraints on the chosen protocols except for the 2^{nd} *and the* 7^{th} *entries which require that the protocol tolerates Byzantine clients and does not need more than 3f+1 replicas to operate. Consequently, the calculation of the discrete matrix D becomes as follows:*

$$\begin{aligned}
D &= \left\lfloor \tfrac{1}{a}.A \;\dot{\vee}\; (e_a - U) \right\rfloor \\
&= \left\lfloor \tfrac{1}{7}. \begin{pmatrix} 1&1&1&1&1&0&1 \\ 0&1&1&1&1&0&1 \\ 0&1&0&0&1&1&0 \\ 0&1&1&1&1&0&1 \\ 0&1&0&0&0&1&1 \\ 0&1&1&1&1&0&1 \\ 0&0&1&1&1&1&0 \\ 0&1&1&1&0&0&1 \end{pmatrix} \;\dot{\vee}\; \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} - \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} \right\rfloor \\
&= \left\lfloor \tfrac{1}{7}. \begin{pmatrix} 1&1&1&1&1&0&1 \\ 0&1&1&1&1&0&1 \\ 0&1&0&0&1&1&0 \\ 0&1&1&1&1&0&1 \\ 0&1&0&0&0&1&1 \\ 0&1&1&1&1&0&1 \\ 0&0&1&1&1&1&0 \\ 0&1&1&1&0&0&1 \end{pmatrix} \;\dot{\vee}\; \begin{pmatrix} 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} \right\rfloor \\
&= \left\lfloor \tfrac{1}{7}. \begin{pmatrix} 7 \\ 7 \\ 6 \\ 7 \\ 7 \\ 7 \\ 5 \\ 7 \end{pmatrix} \right\rfloor = \left\lfloor \begin{pmatrix} 1 \\ 1 \\ 6/7 \\ 1 \\ 1 \\ 1 \\ 5/7 \\ 1 \end{pmatrix} \right\rfloor = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix}
\end{aligned}$$

Thus, according to the values of D, the competition continues between all protocols excluding Q/U and OBFT which correspond to the 3^{rd} *and* 6^{th} *entries, respectively. Then, using the same continuous matrix C calculated in Example 4.10, we get the following evaluation scores:*

$$E = D \circ C = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} \circ \begin{pmatrix} 4.33 \\ 6.01 \\ 3.95 \\ 4.4 \\ 6 \\ 4.18 \\ 8.25 \\ 6.57 \end{pmatrix} = \begin{pmatrix} 4.33 \\ 6.01 \\ 0 \\ 4.4 \\ 6 \\ 4.18 \\ 0 \\ 6.57 \end{pmatrix}$$

Thus, max(E)=6.57=E_5; therefore, the preferred protocol according to these settings is Chain since it corresponds to the last entry in E.

According to the results of Example 4.12, Chain has been chosen as a preferred protocol. However, it is known that Chain achieves low throughput with few clients due to its long messaging pattern (see Figure 3.5). Thus, in case the system is supposed to low contention for a reasonable period of time, it is more efficient to switch to another protocol. In the static mode, Chain could not be replaced by another protocol according to matrix E. However, in the dynamic mode the values of the KPIs in the matrix B can be modified at run time which may yield another protocol that has a better performance.

In Chapter 6, we present a method to assess the values of B on the fly. The method uses the famous Support Vector Machines for Regression [49] (SVR), a technique often used in Machine Learning. This mechanism allows the prediction of the KPIs of the protocols as the system is running. For instance, in Example 4.12, Chain will not be chosen as preferred protocol, in the dynamic mode, if the system state indicates low contention. In fact, the prediction of the throughput of Chain will be low with few clients, which changes the corresponding value in the matrix B; and consequently, the system yields another protocol that achieves better throughput under these new conditions. In addition, this technique leads to very accurate (up to 98%) KPI values which is not the case in our current chapter (as shown in Table 4.5). For example, our experiements show that the throughput of Zyzzyva is very close to Chain in real systems, however the β values of Zyzzyva in Table 4.5 (and in matrix B) are lower than Chain. Indeed, no BFT protocol could preserve its good performance under all condiitions as shown in Chapter 3. We defer the discussion of the dynamic mode to Chapter 6.

4.5.3.2 HEURISTIC Mode

The HEURISTIC mode is an advanced dynamic mode. This mode uses some heuristics to change the user preferences of the KPIs. This is very useful in the cases where the user has no big experience or when the system is fluctuating. Similar to the dynamic mode, the evaluation of protocols is done at run time and, probably, a new protocol is selected as the system state changes. Thus, the KPIs (i.e., the entries of matrix B) are calculated dynamically during the normal operation of the system.

However, in the dynamic mode, it can happen that according to the chosen KPI weights of the user (i.e., the entries of matrix V), a certain protocol is recommended; however, under some specific conditions, other protocols can perform better (thus, the choice is not the best). For instance, if the system is deprived from contention, the latency becomes more important than throughput and capacity, even through the user gave higher weights for these values. Another example is when the system is jammed with requests.

In this case, the system should run the protocol that tolerates more clients even if the client gave low weights to the capacity KPI. The role of the heuristics is to adjust the user preferences through using the weights (in matrix W) as the system state changes. These weights can modify the weights already defined by the user in V, but just temporarily, according to the underlying system conditions.

To explain the idea of heuristic we recall the matrices of the previous example.

Example 4.13. *First, we recall the user preferences of the KCIs U and the KPIs V that were used in Example 4.12:*

$$U = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix} ; and\ V = \begin{pmatrix} 3 \\ 3 \\ 4 \end{pmatrix}$$

In addition we recall the evaluation E in the static mode that was computed in Example 4.12:

$$E \quad = \quad D \circ C = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} \circ \begin{pmatrix} 4.33 \\ 6.01 \\ 3.95 \\ 4.4 \\ 6 \\ 4.18 \\ 8.25 \\ 6.57 \end{pmatrix} = \begin{pmatrix} 4.33 \\ 6.01 \\ 0 \\ 4.4 \\ 6 \\ 4.18 \\ 0 \\ 6.57 \end{pmatrix}$$

According to E, U, and V, the preferred protocol chosen is the one that corresponds to max(E)=6.57=Chain. Now, if the mode is set to HEURISTIC, we consider the following cases.

No contention. In this case, the throughput and the capacity KPIs of the system are not important; on the contrary, the latency becomes the major performance KPI. Thus, the user preferences of the KPIs defined in matrix V=(4,2,4) in Example 4.13 do not reflect the performance of the protocols anymore. Therefore, the weights of these KPIs should be modified using the system heuristics. For instance, the heuristics in this case sets the values of matrix W to:

$$W = \begin{pmatrix} 1 \\ 8 \\ 1 \end{pmatrix}$$

Recall that in the static mode W=e_n. The effect of this modification can be noticed in the following example.

Example 4.14. *Let us calculate the continuous matrix C again.*

$$
\begin{aligned}
C &= B^{\pm}.(V \circ W) \\
&= \begin{pmatrix} 0 & 0.71 & 0.55 \\ 0.25 & 0.86 & 0.67 \\ 0.17 & 1 & 0.11 \\ 0.17 & 0.71 & 0.44 \\ 1 & 1 & 0 \\ 0.06 & 0 & 1 \\ 1 & 0.71 & 0.67 \\ 0.58 & 0.57 & 0.89 \end{pmatrix} . \begin{pmatrix} 3 \\ 3 \\ 4 \end{pmatrix} \circ \begin{pmatrix} 1 \\ 8 \\ 1 \end{pmatrix} \qquad (\textit{step } 0) \\
&= \begin{pmatrix} 0 & 0.71 & 0.55 \\ 0.25 & 0.86 & 0.67 \\ 0.17 & 1 & 0.11 \\ 0.17 & 0.71 & 0.44 \\ 1 & 1 & 0 \\ 0.06 & 0 & 1 \\ 1 & 0.71 & 0.67 \\ 0.58 & 0.57 & 0.89 \end{pmatrix} . \begin{pmatrix} 3 \\ 24 \\ 4 \end{pmatrix} \qquad (\textit{step } 1) \\
&= \begin{pmatrix} 19.24 \\ 24.07 \\ 24.95 \\ 19.31 \\ 27 \\ 4.18 \\ 22.72 \\ 18.98 \end{pmatrix} \qquad (\textit{step } 2)
\end{aligned}
$$

Then, the calculation of E becomes as follows:

$$
E = D \circ C = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} \circ \begin{pmatrix} 19.24 \\ 24.07 \\ 24.95 \\ 19.31 \\ 27 \\ 4.18 \\ 22.72 \\ 18.98 \end{pmatrix} = \begin{pmatrix} 19.24 \\ 24.07 \\ 0 \\ 19.31 \\ 27 \\ 4.18 \\ 0 \\ 18.98 \end{pmatrix}
$$

In this case, max_i=max(E)=27, and thus, Quorum is the preferred protocol since it corresponds to the 5th *entry.*

Example 4.14 depicts the power of the heuristic mode. Notice that though the user requirements are almost equivalent for all KPIs (i.e., throughput, latency, and capacity), in contention-free cases, the heuristics can modify the weights defined in V through multiplying it by W. Consequently, the preferred protocol can change according to these heuristics (as compared to Example 4.13). Example 4.14 shows that the preferred protocol in contention-free cases is Quorum which is very reasonable as it exhibits the shortest messaging pattern among all protocols (see Figure 3.5 in Chapter 3).

High contention. In this case, tolerating a high number of clients is more important than throughput and latency. Again, the user preferences of the KPIs defined in matrix V=(4,2,4) in Example 4.13 do not reflect the performance of the protocols anymore. The weights of these KPIs can be modified using the system heuristics. Now consider the heuristics depicted in matrix W:

$$W = \begin{pmatrix} 2 \\ 1 \\ 7 \end{pmatrix}$$

The effect of this modification can be noticed in the following example.

Example 4.15. *Lets calculate the continuous matrix C again.*

$$\begin{aligned}
C &= B^{\pm}.(V \circ W) \\
&= \begin{pmatrix} 0 & 0.71 & 0.55 \\ 0.25 & 0.86 & 0.67 \\ 0.17 & 1 & 0.11 \\ 0.17 & 0.71 & 0.44 \\ 1 & 1 & 0 \\ 0.06 & 0 & 1 \\ 1 & 0.71 & 0.67 \\ 0.58 & 0.57 & 0.89 \end{pmatrix} . \begin{pmatrix} 3 \\ 3 \\ 4 \end{pmatrix} \circ \begin{pmatrix} 2 \\ 1 \\ 7 \end{pmatrix} \quad (\textit{step } 0) \\
&= \begin{pmatrix} 0 & 0.71 & 0.55 \\ 0.25 & 0.86 & 0.67 \\ 0.17 & 1 & 0.11 \\ 0.17 & 0.71 & 0.44 \\ 1 & 1 & 0 \\ 0.06 & 0 & 1 \\ 1 & 0.71 & 0.67 \\ 0.58 & 0.57 & 0.89 \end{pmatrix} . \begin{pmatrix} 6 \\ 3 \\ 28 \end{pmatrix} \quad (\textit{step } 1) \\
&= \begin{pmatrix} 17.53 \\ 22.84 \\ 7.1 \\ 15.47 \\ 9 \\ 28.36 \\ 26.89 \\ 30.11 \end{pmatrix} \quad (\textit{step } 2)
\end{aligned}$$

And then, the calculation of E is:

$$E \;=\; D \circ C = \begin{pmatrix} 1 \\ 1 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \\ 1 \end{pmatrix} \circ \begin{pmatrix} 17.53 \\ 22.84 \\ 7.1 \\ 15.47 \\ 9 \\ 28.36 \\ 26.89 \\ 30.11 \end{pmatrix} = \begin{pmatrix} 17.53 \\ 22.84 \\ 0 \\ 15.47 \\ 9 \\ 28.36 \\ 0 \\ 30.11 \end{pmatrix}$$

In this case, max_i=max(E)=30.11, and thus, Chain is the preferred protocol since it corresponds to the last entry.

Again, Example 4.15 presents the modification of user requirements according to system heuristics. The preferred protocol under high contention becomes Chain. This is expected since Chain is deprived from collisions under contention due to its chain-messaging fashion.

The above examples are conducted without changing the values of KPIs in matrix B (i.e., as if no dynamic mode is chosen). Indeed, this makes the result a bit biased. The reason is that, under high contention, Ring achieves a very high throughput whereas the other protocols go in a plateau as observed in Figure 4.4. However, the matrix B uses the throughput 0.06 of Ring, which is very different from reality. The issue arises here since we estimated the throughput of Ring according to the number of MAC authentications needed which could not be true as the system state changes. This indicates that when the KPIs in B are calculated dynamically, the results can be more accurate and reasonable (discussed in Chapter 6). For this example, Chapter 6 shows that Ring will be the best choice using the prediction system of KPIs. This sounds very logical since the throughput of Ring will be much higher than 0.06 (i.e., the estimated value used in this example).

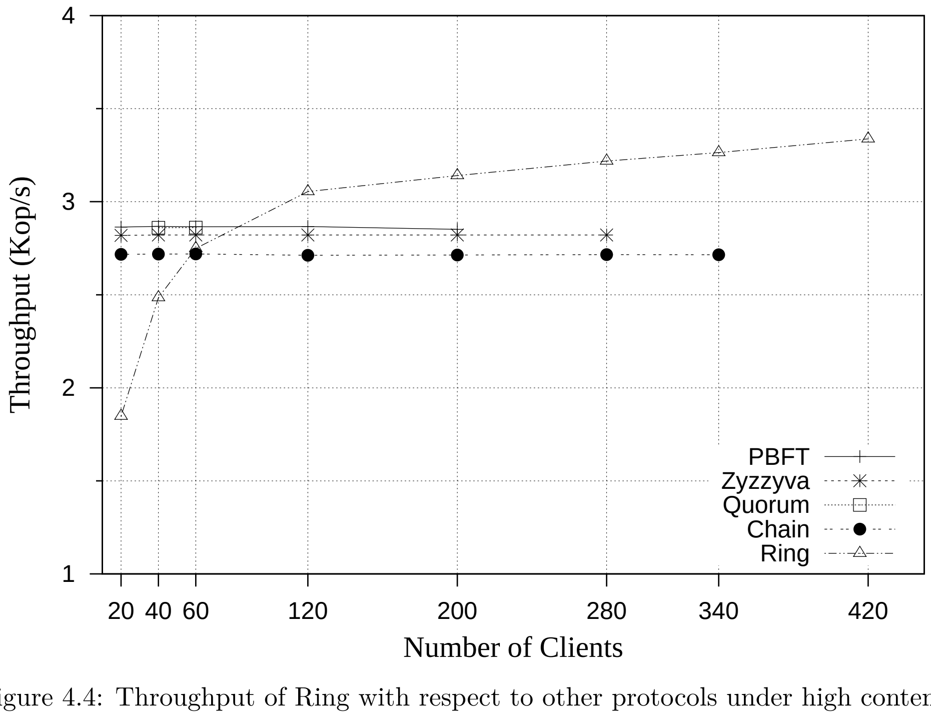


Figure 4.4: Throughput of Ring with respect to other protocols under high contention.

4.6 Worthy Switching

In the modes DYNAMIC and HEURISTIC, the system switches to a new protocol that is evaluated as preferred protocol under the new system state. Switching takes place either during a routine system performance checking and evaluation, or upon receiving a failure event. Worthy switching will be discussed in more details in Chapter 6.

Routine Checking. In this case, the system checks the performance of the system periodically. This is done through performing the evaluation process discussed in this chapter, periodically, at a certain predefined duration. After that, the system can switch to another protocol seeking better performance. However, if the difference in performance between the running protocol and the next one is 'small', switching becomes useless. Instead, we require that switching to occur only if:

$$E_{max} - E_{curr} \geq S_{thr} \tag{4.7}$$

E_{max} and E_{curr} in Equation 4.7 are the evaluation scores corresponding to the preferred protocol (under the current state s_t) and the currently running one, respectively. S_{thr} represents the threshold over which switching is considered worthy. S_{thr} is a decision choice of the service administration or the user (e.g., 10%).

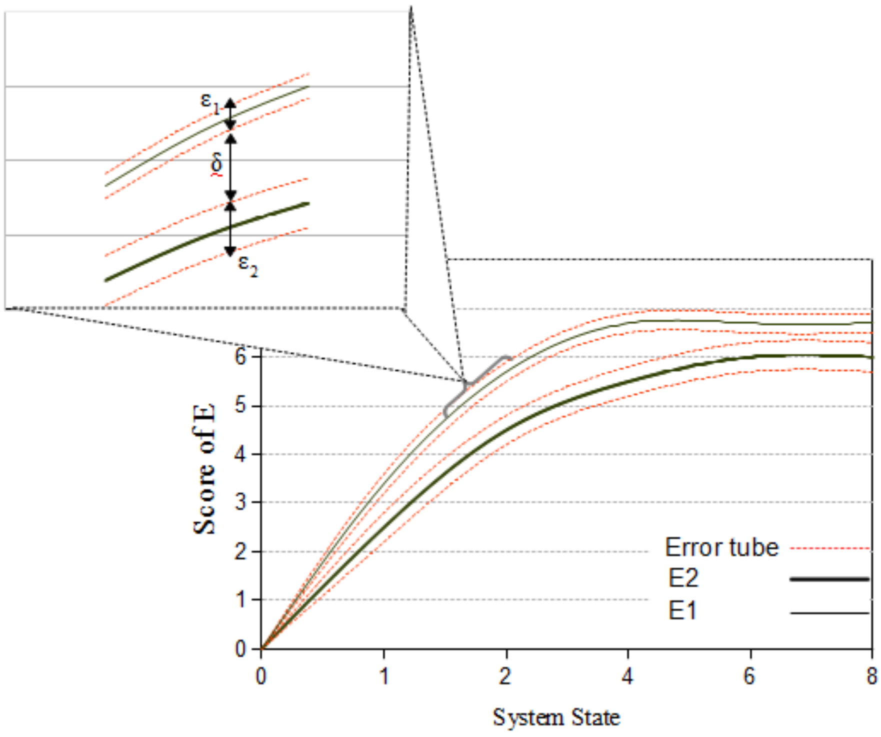


Figure 4.5: The difference between evaluation scores (E) with considering the error tubes.

Figure 4.5 conveys a sample graph of the evaluation scores of two different protocols as the system state changes. Since E depends on predicted values of KPIs, then each E has an average prediction error represented by the error tubes on the graph (in red color). For system state s_t=2, we plot a magnified graph to show the differences between E1 and E2. ϵ_1 and ϵ_2 represent the width of the error tubes of E1 and E2, respectively. δ is the distance between E1 and E2 after excluding the width of the error tubes. Switching is worthy only when the switching threshold S_{thr} is less than or equal to δ, in other words:

$$E1 - E2 - \epsilon_1/2 - \epsilon_2/2 = \delta > S_{thr} \tag{4.8}$$

Notice that, ϵ_1 and ϵ_2 should be excluded to ensure that, in worst cases, the difference δ between E1 and E2 is always guaranteed. This is needed to avoid the overhead of switching if the prediction is not accurate enough. We give more discussion and examples in Chapter 6 about this point.

Abort. In addition, switching can occur when something wrong happens (e.g., a Byzantine replica detected). An evaluation process takes place before switching. In this case, the system sets the matrix U of Equation 4.4 to e_n; this forces any speculative protocol that can not work under replica failures to be out of the competition (e.g., Quorum). Therefore, the selected protocol for the next phase will be the one that can tolerate replica failures, and at the same time, the one that has the preferred performance among all protocols.

4.7 Conclusion

In this chapter, we presented a BFT selection model and algorithm to improve the quality of BFT services. The mechanism helps the user to choose the 'preferred' BFT protocol according his preferences. This is useful in large services that provide BFT as a service. For instance, cloud vendors can sell BFT services along with their IaaS, PaaS, and SaaS SLA contracts. This can be achieved by proposing many BFT protocols for users, and the user is in charge of choosing his preferred protocol. This can be implemented as a Web service.

Our mechanism tries to match the user preferences with the profiles of each protocol through a set of mathematical formulas based on matrices. The formulas are automated to choose the preferred protocol with respect to the demanding user. Our model considers three modes: (1) STATIC mode: this is the default mode where the user chooses a protocol only once; he can only change it when the service is rebooted. (2) DYNAMIC mode: which allows the user to use more than one protocol at once, where a running protocol can be stopped and another protocol is launched after performing the evaluation and selection process. The intuition is that the performance of protocols differ as the underlying system state changes, and thus adaptation to the new state is required. In this mode, the performance measures (i.e., the KPIs) are calculated at run time (to be introduced in Chapter 6). (3) HEURISITC mode: this mode is similar to the dynamic mode; however, it allows to modify the weights (i.e., preferences) chosen by the user as the system state changes using some predefined heuristics. This mode is very useful since some KPIs are important under some conditions, but they become less important in other situations. For example, throughput is very important in normal cases, however, it is not basic in contention-free cases (in contrast to latency), thus, adjusting the weights of the KPIs will have great impact on the overall performance.

In the next chapters, we present some BFT techniques that can be used in the STATIC mode. In addition, we provide Machine Learning prediction method to assess the KPI values of the protocols, while the system is running, using Support Vector Machines for Regression (SVR) [49]. In the future, we are planning to study the cost of this selection mechanism and to explore the heuristics required for the heuristic mode.

Chapter 5

BFT Reconfiguration: BFT Re-Abstract Family

This chapter presents a technique to improve the performance of BFT protocols in the static mode that we discussed in the previous chapter. The idea is to conserve the performance of any protocol even after recovery. This approach allows any protocol, that is a member of the Re-Abstract family, to recover to itself upon failure, and hence, avoiding usual expensive recovery phases. The chapter provides some examples on three members of this family: Zlight, Quorum, and Chain.

5.1 Introduction

Many BFT protocols ([15, 2, 22, 32],...) have been designed in the last decade; however, none of them fits the wide range of infrastructures and settings of the deployed services (as discussed in the previous chapters). PBFT [15] was the first practical BFT protocol to appear. PBFT suffered from the low performance despite being one of the most robust BFT protocols. The speculative approaches [2, 22, 32, 26] then succeeded to ameliorate the performance during the failure-free phases, but a recovery phase was crucial to resolve the issues under failure. The recovery phase in such protocols render a low performance, similar to that of PBFT or even worst in some cases like Zyzzyva, where PBFT performs better than this latter by 15% under failure [32]. Consequently, those speculative protocols were not satisfactory to services that seek a stable performance even during recovery, especially if the failure is not transient.

The abortable platform, i.e. *Abstract*[1] [27], then introduced the possibility of adapting to new conditions that might show up in a deployed services. The modular nature of *Abstract* allowed the recovery from one *Abstract* (any BFT protocol) to another, according to the upcoming changes in the service. *Abstract* platform addressed the trivial (failure-free) cases; however under faults (1) it recovers to PBFT-like backup that exhibits a low performance; or (2) it launches another protocol. However, no dynamic switching policy was defined for this approach. More details about abortability can be found in Chapter 2 and in [27, 26].

We present in this chapter a family of modular BFT protocols, called *Re-Abstract*, built on top of *Abstract* platform; where *Abstract* can be any traditional protocol. The idea is to have the *Abstract* (i.e, any protocol) recover back to itself upon failure detection

[1]The term 'Abstract' comes from: ABortable STate mAChine Replication

(instead of switching to PBFT-like backup, as done in [27]). *Re-Abstract* family members conserve their original inherited performance even during recovery. This is done via the *recovery plug-in*, that we built to detect and replace the faulty replica by another correct backup replica. We benefit from the *Abstract platform* to make the protocols simple to re-use, design, and implement.

As an example on Re-Abstract application; consider a deployed service that should be secure and fault resilient, but it does not experience a high contention usually. Such a service recommends a protocol with a good latency, like Q/U [2] or Quorum [27], since throughput does not matter in contention-free environments. Upon failure, the latency of Q/U becomes very large due to the expensive *Repair* phase. On the other hand, Quorum recovers to a PBFT-like protocol, which also, in its turn, has a high latency. A more convenient solution is to preserve the same latency even after recovery through using *Re-Quorum*; thus, by recovering from Quorum to Quorum itself, after replacing the faulty replicas. By this, the anticipated performance of the deployed service can be always maintained. Figure 5.1 for instance depicts the throughput of some state of the art BFT protocols when experimented on Redis software. The figure clearly highlights the difference in performance among protocols, especially between PBFT and Quorum. Imagine the costly overhead that would happen upon recovery of Quorum to PBFT; this can be avoided upon recovery to Quorum (i.e., to itself) again using our approach.

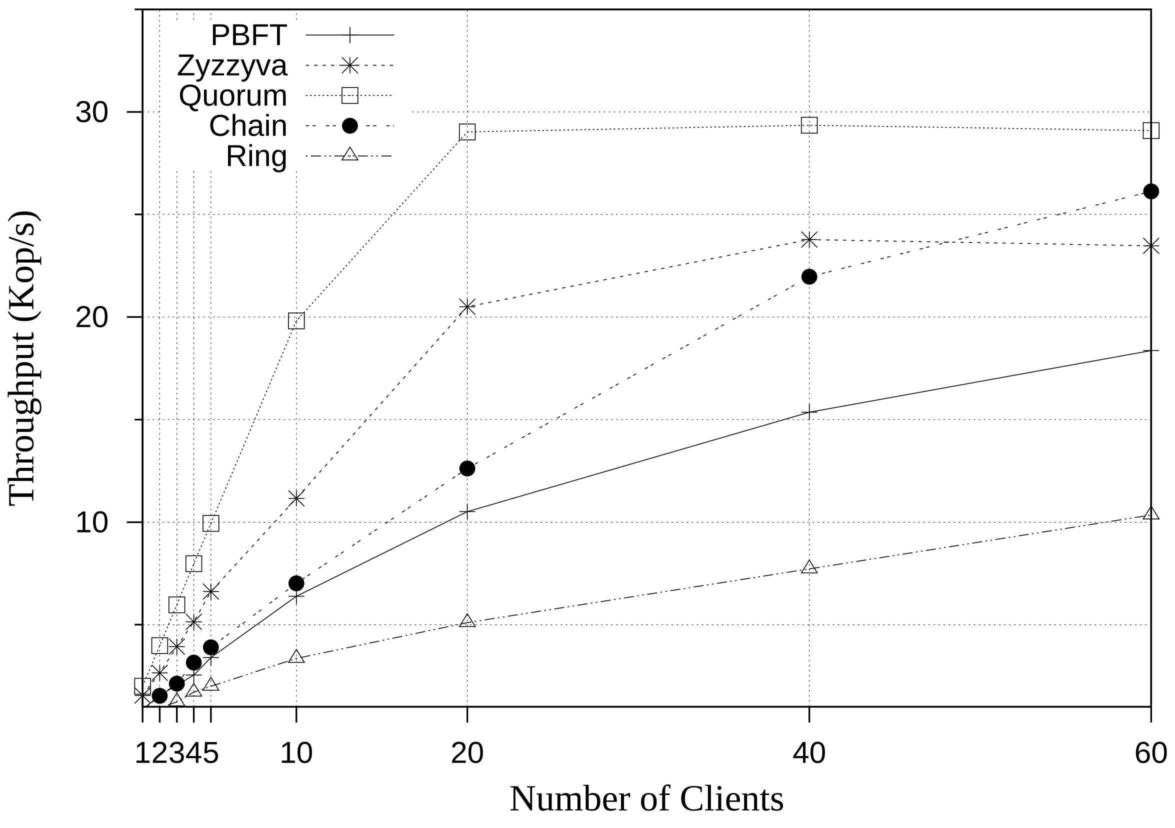


Figure 5.1: Throughput using the 0K payloads on Redis.

In brief, Re-Abstract family members work as follows. For simplicity, we consider one *Re-abstract* family member: Re-Zlight [2]. System replicas are composed of two sets: an

[2]Zlight is an abortable version of Zyzzyva built using the *Abstract platform*

Active set ($3f$+1 replicas), and a *Passive set* (f backup replicas). *Re-Zlight* operates on the *Active replicas* during its *Speculative phase*, according to its defined message pattern (see Figure 5.3). Once a client detects a failure, it initiates the *Recovery phase* to identify and eliminate the potentially faulty replicas (they can be just slow). The client replaces the eliminated replicas by backup replicas from the Passive set. Re-Zlight continues then through its speculative phase following the usual message pattern again, just after the switching phase finishes. This way, the service maintains Zlight's performance even after recovery.

Re-Abstract family maintains one-copy semantics, and conserves a constant service performance. However, Re-Abstract family members require f extra *Passive* replicas as compared to state of the art protocols ($3f$+1). This is tolerable nowadays, as commodity hardware and *cloud computing* solutions are becoming cheap. Re-Abstract imposes some additional, but negligible, cost while recovery.

5.2 Design of Re-Abstract

In order to make the presentation simpler, we present *Re-Abstract* family through discussing *Re-Zlight*. Knowing that any traditional BFT protocol can be a member of the family.

5.2.1 Fault Model

Here we present our fault model which complies with the traditional BFT model (e.g., PBFT [15]). We assume a message-passing distributed system using a fully connected network among nodes: clients and servers. The network may (not infinitely) fail to deliver, corrupt, delay, or reorder messages. Faulty replicas and clients may either behave arbitrarily, i.e., in a different way to their designed purposes, or they just crash (*benign* faults). A strong adversary coordinates faulty replicas to compromise the replicated service. However, we assume the adversary cannot break cryptographic techniques like: collision-resistant hashes, encryption, and signatures. Liveness, however, is guaranteed only whenever the system is *eventually* synchronous; i.e., during intervals in which messages reach their correct destinations within some fixed worst case delay.

5.2.2 Algorithm Overview

Re-Zlight requires at least $4f$+1 replicas to tolerate up to f Byzantine replicas and any number of Byzantine clients. However, using $3f$+1 replicas only at a time, it can sustain faults, but cannot ensure progress. At any time, the protocol distributes the replicas into two sets:

- *Active* set: composed of $3f$+1 replicas (we call them *Active* replicas); these replicas are used in the *speculative* phase, speculatively.

- *Passive* set: composed of f idle replicas that are used as recovery backups.

Re-Zlight launches the *speculative* phase on $3f$+1 *Active* replicas. Upon failure detection, it recovers by replacing the *suspicious* replicas (i.e., either faulty or slow) by correct replicas from the f *Passive* set; and then, resumes to the speculative phase on a new Active set (i.e., in a new view). The $3f+1$ replicas are enough to collect a correct *abort history.*

The algorithm of Re-Zlight consists of two main phases: a *speculative* phase and a *recovery* phase. In this section, we recall the two phases briefly, and then we only present the recovery phase in details (since the speculative phase is similar to Zyzzyva's trivial case discussed in Chapter 2 and in [32]).

Speculative phase. The communication message pattern of Re-Zlight in a failure-free scenario is simple (see Figure 5.3), and it is concerned with the Active set only (i.e., $3f+1$ replicas):

1. The client sends its request to the *primary* replica.

2. The primary assigns a sequence number to the request and executes it.

3. The primary sends the reply to the client, and forwards another copy to the remaining $3f$ Active replicas, along with the assigned sequence number.

4. Each non-primary replica executes the received requests by order, and returns the replies to the client.

5. The client commits the request only if all the responses of the Active replicas match; otherwise, the recovery phase is launched.

Recovery phase. This phase takes place using both: Passive and Active sets (i.e., $4f+1$ replicas).

1. Once the timer of the client expires waiting for $3f+1$ matching replies, the client panics by sending a *Panic* message to the Active replicas.

2. Replicas, upon receiving the Panic messages, stop executing new requests and send an *Abort* message back to client with their signed *local histories.*

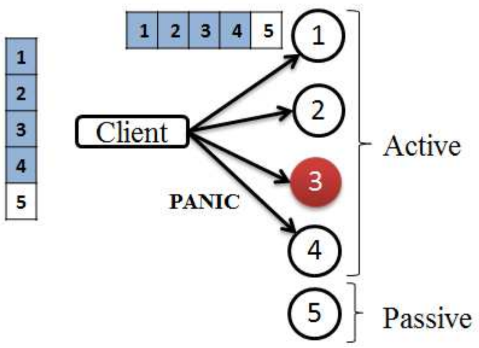
3. The client constructs an *Abort history* collected from the $2f+1$ matching replies (more details later), and sends an $INIT$ request to all replicas ($4f+1$) along with the abort history.

4. The replicas execute the $INIT$ request (they append it to their local histories), and reply to the client with ACK_{init}.

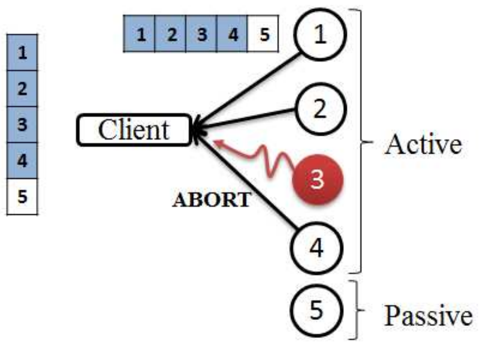
5. As the client receives the first $3f+1$ matching ACK_{init} replies, it considers their corresponding replicas as Active, and the remaining f replicas as Passive (those are *suspicious* replicas).

6. The client sends SET message to update the Active set on the replicas with its request piggybacked.

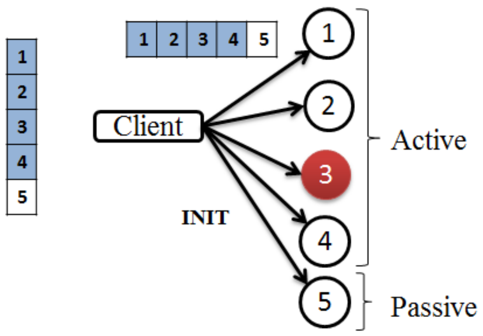
7. All replicas update their local control information (i.e., their Active set), and Active replicas reply to the client according to the speculative phase in a new view.



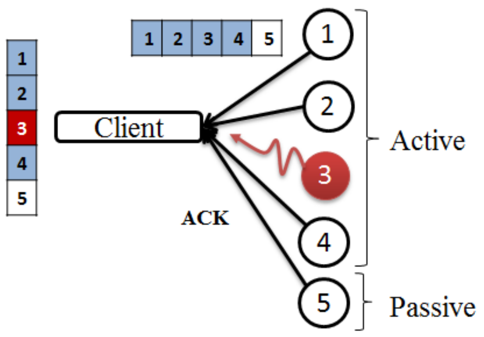
(a) Client *c* sends a message to Active set. CONTROL structure of replicas and clients are the same.



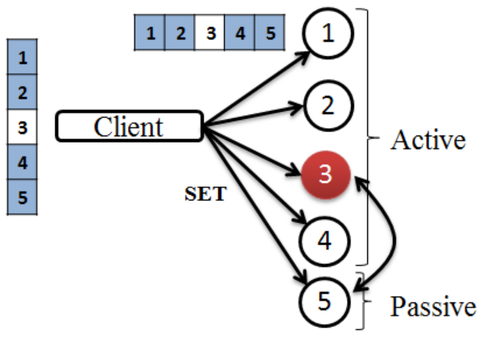
(b) Active replicas respond. Replica three might be Byzantine.



(c) Client *c* detects a faulty replica sends INIT message to Active and Passive replicas to replace the faulty one.



(d) Replicas reply with ACK message; the client detects that replica 3 is suspicious (Byzantine or slow)).



(e) The client broadcasts the new info. to update the CONTROL structure of replicas.

Figure 5.2: The mechanism used to evict the faulty replica from the Active set. The tables with numbers are the CONTROL structures of the client and replicas, respectively.

5.2.3 Details of the Recovery Phase

The *Recovery* phase is composed of three major steps: aborting, collecting abort history, and cleaning the *Active* set from any *suspicious* replicas. Figure **??** explains the mechanism in brief.

Aborting. The client in Re-Zlight considers a request as complete if the received $3f+1$ responses of the Active replicas are matching and before the expiry of the timer. Otherwise, the client stops sending requests and sends a *Panic* message to all Active replicas. Each replica, upon receiving the Panic message, stops receiving/executing requests, appends its *local history* of committed requests to an *Abort* message, and sends it to the client. The latter waits until it receives $2f+1$ non-conflicting signed Abort messages. This ensures the correctness of the local histories. Aborting is achieved as follows:

1. The client waits for the first $2f+1$ local commit histories to be received.
2. If no conflicting entries among the $2f+1$ received local histories are found by the client, it stops receiving new histories, and collects the $2f+1$ messages in a $Proof_{AH}$ set, that is used to form the abort history AH later.
3. Otherwise, if the client identified some commit histories with conflicting entries it waits for new local histories (since definitely there are correct clients that did not respond yet). The loop continues from step 1 again.

Building Abort History. A correct *abort history* is crucial for safety. It preserves total ordering and consistency across different switching phases (i.e., views). The abort history is collected from the current Active set, to initialize the local histories of the replicas on a new correct Active set. Building the abort history AH is done by the client after the receipt of $2f+1$ non-conflicting signed abort messages from different replicas, collected in the $Proof_{AH}$ structure. The steps can be summarized as follows:

1. The client generates a history h such that: $h[j]$ equals the value that appears at position $j \geq 1$ of $f+1$ different local histories (LH_j), that appear in $Proof_{AH}$.
2. If such a value does not exist for some position x, then x is the last index of h.
3. Finally, AH is the longest prefix of h in which no request appears twice (i.e., exclude duplicate entries).

AH is used, then, to initialize the local histories of the new Active replicas (see more details later).

Eliminating Faulty Replicas. The client in Re-Zlight attempts to replace the faulty replicas in the Active set with correct ones from the Passive set; this is done by detecting the suspicious replicas (faulty and slow ones) among the $4f+1$ replicas. This occurs as follows:

1. After collecting the abort history AH, the client sends an $INIT$ request (with AH appended) to the $4f+1$ replicas.
2. The replicas execute the INIT request, and append AH to their local history (if it is not already done through a previous view). Then, they reply to the client with ACK_{init}.
3. The client verifies the received ACK_{init} messages, and waits until it receives $3f+1$ matching ones.
4. The corresponding replicas to the first $3f+1$ matching messages constitute the new Active set, and the remaining f form the Passive set.

This process ensures eliminating the faulty replicas from the Active set, and having a correct one in the next view. Note that, all the exchanged messages among the replicas and the clients are authenticated/verified with MACs.

5.2.4 Switching

Switching requires $4f + 1$ physical replicas where only $3f + 1$ of them are used at once as Active replicas. The other Passive replicas, however, are used to replace some Active replicas whenever failure occurs. This requires, sometimes, changing the primary replica also.

Initializing control information on clients. Upon launching the system, default Active set and primary replica are chosen. And since these are changing among the same physical replicas across different views, it is crucial for the client to know this *control information* before issuing its requests. The control information is collected on replicas and clients in the $CONTROL$ structure, that is composed of: (1) The Active set $ACTIVE$, (2) the primary P, and (3) the current view number $VIEW_{curr}$. Hence, some control messages are needed to deliver this information to any new coming client. The client gets informed as follows:

1. The client sends a GET_{info} message to all $(4f + 1)$ replicas.

2. *Active* replicas send an $INFO$ message to the client containing the control information: (1) The Active set $ACTIVE$, (2) the primary P, and (3) the current view number $VIEW_{curr}$.

3. The client waits until it receives $2f+1$ matching $INFO$ messages from the replicas. Once done, it saves the information of the $INFO$ messages, updates its CONTROL, and starts sending its requests according to the protocol. Otherwise, it starts again from step 1.

Control information after recovery. However, when recovery occurs, and some client c succeeds in panicking, sending the $INIT$ request, and identifying the suspicious replicas (faulty or slow), the control structure CONTROL should change to exclude faulty replicas. Thus, the replicas refuse to execute any request until: a SET_{info} control message (more details later) is received from c, or the timer expires. Instead, the replicas respond on any request type with an $INFO$ message (possibly with some empty fields).

In the case where the timer of a replica expires, it releases the lock, and accepts Panic messages in the current view to allow other clients to perform a successful recovery. SET_{info} is important after recovery by which replicas get assigned a new CONTROL. To fill the SET_{info} message, the client determines the Active set, selects the first replica to be the primary, and increments the view number. The client appends $Proof_{AH}$ to the message so that the replicas verify message correctness against malicious clients.

If CONTROL on the replicas is already updated by the SET_{info} message, the INFO reply message will be complete and informative enough for other clients (upon receiving $2f+1$ matching messages). Otherwise, if the replicas are still waiting the SET_{info} message from c, the ACTIVE and the primary replica P fields in the INFO message will be empty, in contrast to the $VIEW_{curr}$ field that allows the clients to retry sending Panic messages in the current view successfully; since replicas do not accept any requests from other views. Re-Zlight handles SET_{info} messages according to the usual communication pattern in (Figure 5.3). The receiving replica knows from the SET_{info} message itself, whether it is included in the Active set, it is a new primary, or it is a Passive replica; nd it can verify this using the $proof$ message.

5.2.5 Handling Contention

Views Perspective. Requests of any type are required to comprise the current view number $VIEW_{curr}$. In addition, any request received by the replicas is validated by verifying its MACs and the $VIEW_{curr}$ field, before being executed. Any request that belongs to a different view or has corrupted MAC gets rejected.

Panicking. The speculative phase in Re-Zlight is deprived from contention problems as long as all requests that belong to the current view are ordered by the primary. However, upon failure, a client launches the recovery phase. The client needs to collect the abort history. Correct clients keep retransmitting *Panic* messages until they receive an enough number (i.e., $2f + 1$) of matching local histories, or a filled INFO messages from the replicas.

Initializing. When any client (one or more) creates the abort history, it sends $INIT$ message to all the replicas ($4f + 1$) to initialize their local histories and to update their view number $VIEW_{curr}$. Under contention, different replicas might receive different $INIT$ requests from different clients, and hence none will be completed since no client will be able to collect enough ACK_{init} from $3f+1$ replicas. Thus the clients follow an exponential back-off scheme that offers more chance that all replicas execute the same $INIT$ requests sent by the same client c. This ends up by consistent local histories on replicas in the next view.

Control Information. Afterwards, the replicas will be waiting for the SET_{info} request from the same client c. During this period, and to maintain non-conflicting messages from contending clients; any request will be discarded by the replicas, that reply instead, by an $INFO$ message (where its Active set and primary replica fields are still empty), containing the $VIEW_{curr}$ that is needed by the clients while retrying their future request attempts. The step ends with a unique Active set and a primary replica across different views once the replicas receive the SET_{info} message from the client c. After this stage, any request from the clients will be handled as designed if it belongs to this view, otherwise, replicas respond with an $INFO$ message (where no fields are empty this time) to update those clients with the new control information. By this way, the clients can send requests as usual in the new view.

5.2.6 Byzantine Clients

In this section, we focus on the behavior of malicious clients upon recovery. For the trivial cases of the protocols, i.e., the speculative phase, we do not provide a discussion since this is done for every protocol aside. In fact, most previous works of existing protocols introduced this discussion.

In the recovery phase, a client starts by broadcasting a Panic message to all replicas. In this case, a malicious client can send different messages to different replicas. The replica that receives a Panic message aborts and stops handling new requests from other clients, but it can respond with an INFO messages instead. Whereas, other replicas respond correctly to correct clients. Correct clients then Panic again since it could not receive matching replies from all replicas. The replicas (except the already aborted one) then send an Abort message to the client accompanied with their local histories LH. The already aborted replica aborts again (due to the new Panic message from the new client) after its timer expires while waiting for the INIT message from the first client. Recall that, an exponential back off scheme is applied here to fight against contending clients.

When the replicas send Abort messages to some client, and the client is malicious, it can either drop these messages or respond. In the former cases, the replicas wait until their timer expires waiting for the INIT request from this clients, and then other correct clients panic again and the recovery process continues correctly with the new clients.

The latter case where the client responds, it can respond with a forged INIT message. Correct replicas could not be fooled by this forged message since they can verify the $Proof$ message in the AH that contains their collected and signed local histories LH. Thus, correct replicas drop the INIT message and wait for another correct client to resolve the situation.

If a malicious client sent a correct INIT request, and the replicas reply back with ACK messages. The client can drop this ACK message. Once more, the replicas accept Panic messages from new clients after their timer expires while waiting for Set request for the first client.

When the client receives the first $3f+1$ ACK messages from correct replicas, it designates the corresponding replicas as Active and sends a SET message to the replicas to update their CONTROL information. A malicious client can pretend that some Abort replies arrived before even if the converse is true. By this, the client tries to include slow replicas in the Active set. The impact, however, is limited to slowness and could not affect correctness. Moreover, if the client sent different Set messages to different clients, correct replicas discover this since through calculating themselves the CONTROL information from the $Proof$.

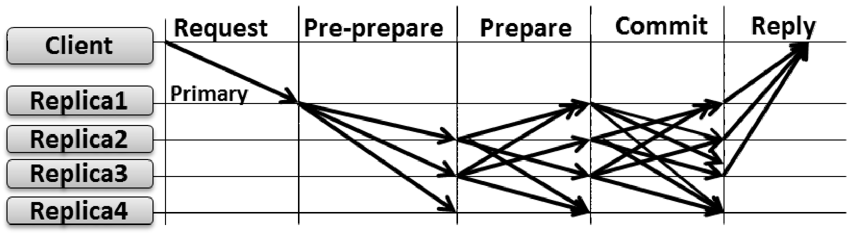
Notice that, malicious clients can delay their replies until just before the expiry of the replicas timer. By this the client succeeds to impose an additional response delay on request. In addition, similar to existing protocols, malicious clients can play a Denial of Service attack. A black list of clients and multiple NIC interfaces on replicas, then, could be used as in [19]. These issues, however, affect performance and have no impact on correctness.

5.3 Evaluation

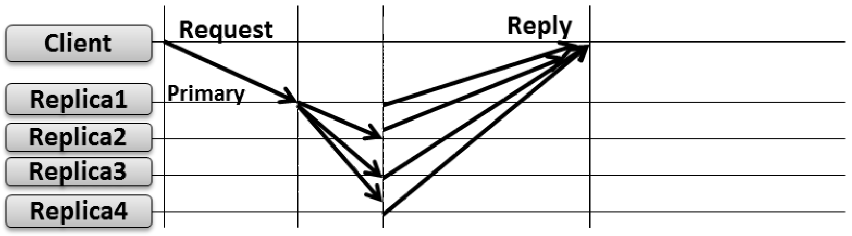
In this section we show the gain in performance, more specifically throughput, upon using our approach. Before starting our discussion, we first introduce the experimental settings we used to conduct our experiments.

5.3.1 Experimental Settings

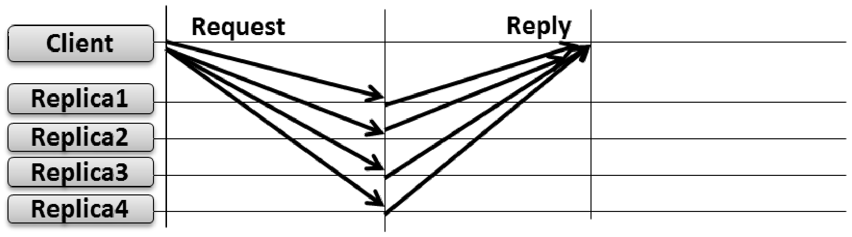
We conducted our experiments using Redis and OpenLDAP. The two software have operations of different execution times which is important in our study as will be shown later. The setup of the system is done on Emulab [55] testbed, where we used *Dell d710* machines – 64-bit Quad Core Xeon systems, with an eight-core 2.4GHz processor, 12 GB of RAM, and 2 NICs. At Emulab, each machine in our experiments runs Ubuntu 8.04, with 2.6.24 32-bit kernel. Replicas are systematically running on their own, separate machine, while clients are collocated on a total of 20 machines. Moreover, in all experiments we use 4 replicas (which consists in tolerating $f=1$ fault).



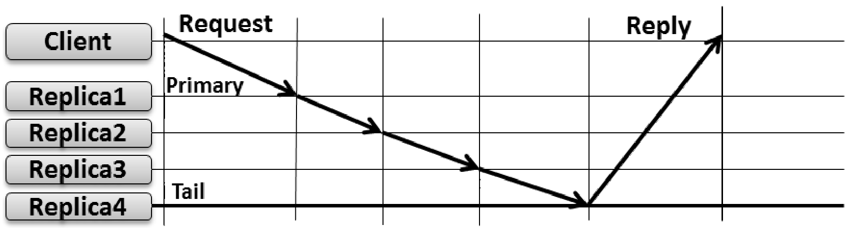
(a) PBFT



(b) Zlight (Zyzzyva)



(c) Quorum



(d) Chain

Figure 5.3: Message patterns of PBFT, Zlight, Quorum, and Chain (for $f = 1$).

5.3.2 Switching Overhead

Figure 5.4 conveys the switching cost of Re-Zlight over LAN for $f = 1$. The figure depicts the recovery time in (ms) as a function of the number of requests in the abort history AH. The size of requests used is 1KB (this size is an average for the typical request sizes used in previous works, e.g., [15, 32, 27, 26], etc). The graph shows that as the *abort history* AH size increases, the recovery time also increases, ranging from 28ms and 42ms. For a 300 request-size of AH, the recovery time does not exceed 50ms. We consider this cost to be very reasonable, provided that faults are supposed to be rare in environments that run speculative protocols like Re-Zlight, Re-Quorum, etc.

We mention here that we did not consider AH of sizes more than 300 requests since, in speculative protocols, fine-grained checkpointing is usually used to reduce the recovery cost (since the abort history AH will be smaller). On the other hand, since the switching cost of all protocols is almost similar due to the slight difference in their message headers, we do not reveal all the graphs of these protocols (in fact, we did not implement the

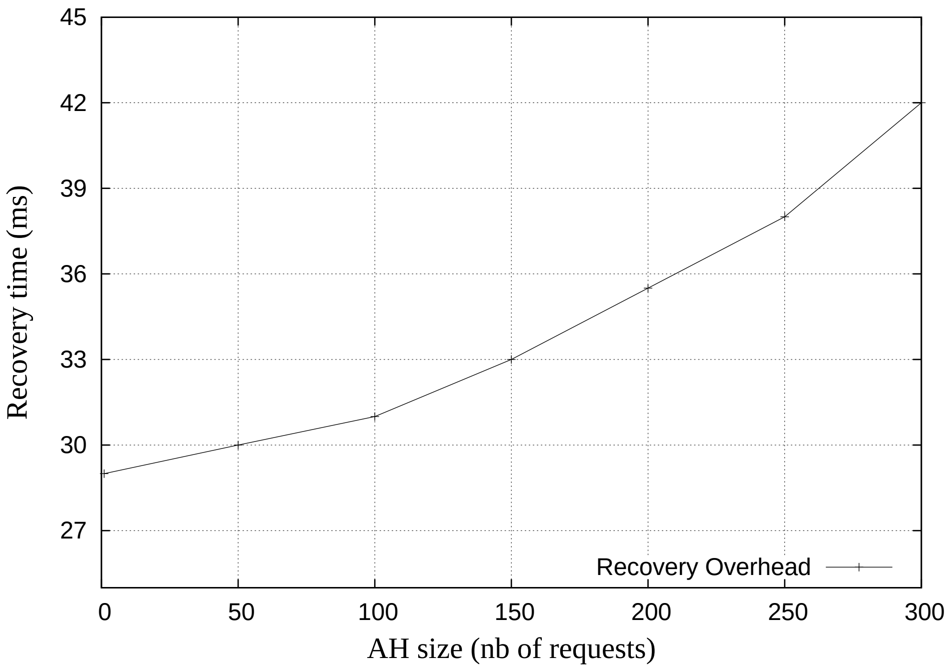


Figure 5.4: Re-Zlight Recovery cost as a function of abort history size, for f=1.

switching phase of all protocols being very similar).

On WAN settings, the execution time of requests becomes negligible as compared to the end-to-end latency (typically, more than 10ms). Thus, the major factor becomes the number of communication steps needed to recover. Table 5.1 shows the number of communication delays needed for state of the art protocols during the speculative phase; and conveys the recovery cost of Re-Zlight, Re-Quorum, and Re-Chain. We show that, the cost during recovery is always double that of the message cost during the speculative phase. Thus, even in WAN settings, the recovery cost is considered small too.

Protocol	PBFT	Re-Zlight	Re-Quorum	Re-Chain
Message delays	4	2	2	2
Recovery delays	-	4	4	4

Table 5.1: Recovery cost of Re-Abstract family members in WAN setting.

5.3.3 Performance

The goal of using our approach is to use any existing protocol as an abortable protocol making it able to recover to itself; this yields a high performance as compared to the original protocol upon recovery. We recall that PBFT achieves the best throughput under failures, whereas other protocols either use a PBFT-like recovery phase (e.g., Quorum, Chain, Zlight [26]) or other techniques that are more expensive (e.g., Zyzzyva [32] and Q/U [2]). Consequently, we focus our comparisons on PBFT as a recovery protocol; and on the trivial cases (i.e., failure-free) of other existing protocols: Zlight, Quorum, and Chain.

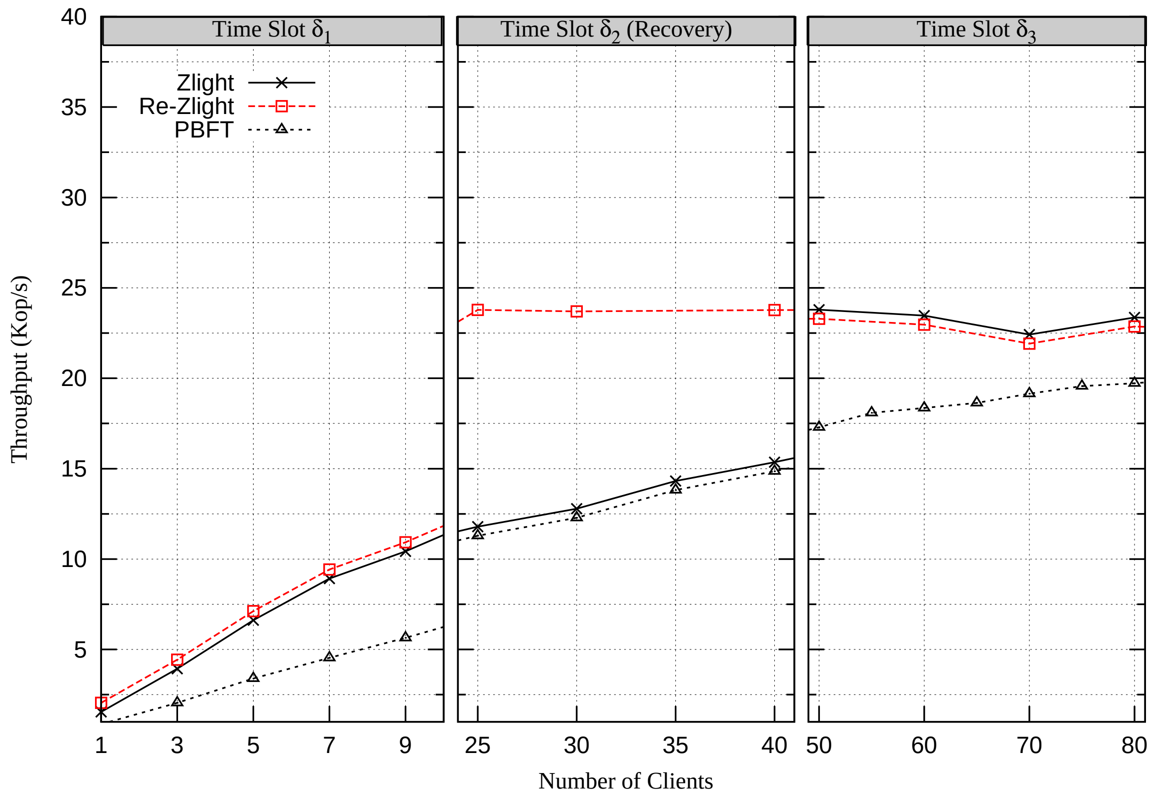


Figure 5.5: Throughput of Re-Zlight during three time slots.

To make our evaluation easier, we consider the following scenario. Assume we have a running BFT service that runs a protocol for a duration δ_1. Then, assume that as the number of clients increases, some failure happens and recovery is launched. The recovery phase (or recovery protocol) runs for a duration δ_2 until the faulty replica is fixed. Then, the protocol switches back to the non-faulty phase which has a better performance as compared to the recovery phase, the protocol runs in this phase for a duration of δ_3. To make the graphs easy to follow, we assume that the number of clients is always increasing with time. In our evaluation, we try to calculate the number of operations performed by each protocol in the phases $\delta = \delta_1 + \delta_2 + \delta_3$, of course, taking into consideration the overhead of switching time. Hence, it is imperative to start with the switching cost.

5.3.3.1 Throughput

In this section we study the throughput of Re-Abstract family members, we consider three protocols: Zlight (i.e., a light version of Zyzzyva), Quorum, and Chain. Any other protocol can be discussed similarly.

Re-Zlight. Figure 5.5 conveys the throughput (on the y-axis) of PBFT, Zlight and Re-Zlight as a function of the number of clients (on the x-axis). The graph is divided into three time periods δ_1, δ_2 and δ_3. In this figure, we notice that the throughput of PBFT is always worst than Zlight, which is expected due the high number of MAC authentications needed in PBFT and due its long message pattern (see Figure 5.3). On the other hand, the throughput of Zlight increases gradually as the number of clients increases up to 10 clients, where the throughput becomes almost double that of PBFT's throughput. In time slot δ_2, we suppose that Zlight switches to PBFT due to some failure, thus the throughput of Zlight drops to almost half its expected throughput. The throughput returns to increase as the failed replica gets repaired in δ_3 and Zlight is able then to run its speculative phase. The throughput drops slightly with more than 70 clients but it remains better than PBFT. Therefore, the throughput of Re-Zlight is similar to Zlight during δ_1 and δ_2, however, during δ_3 it recovers to itself again (by replacing the faulty replica) which conserves its expected performance that is double that of PBFT.

Protocol	PBFT	Zlight	Re-Zlight	Quorum	Re-Quorum	Chain	Re-Chain
δ_1	3K	7K	7K	10K	10K	4K	4K
δ_2	13K	**13K**	24K	**13K**	28K	**13K**	17K
δ_3	18K	23K	23K	27K	27K	26K	26K
$\delta_1 + \delta_2 + \delta_3$	34K	43K	54K	50K	65K	43K	47K
% of gain	-	-	20%	-	23%	-	10%

Table 5.2: The average number of operations performed in the each time slot.

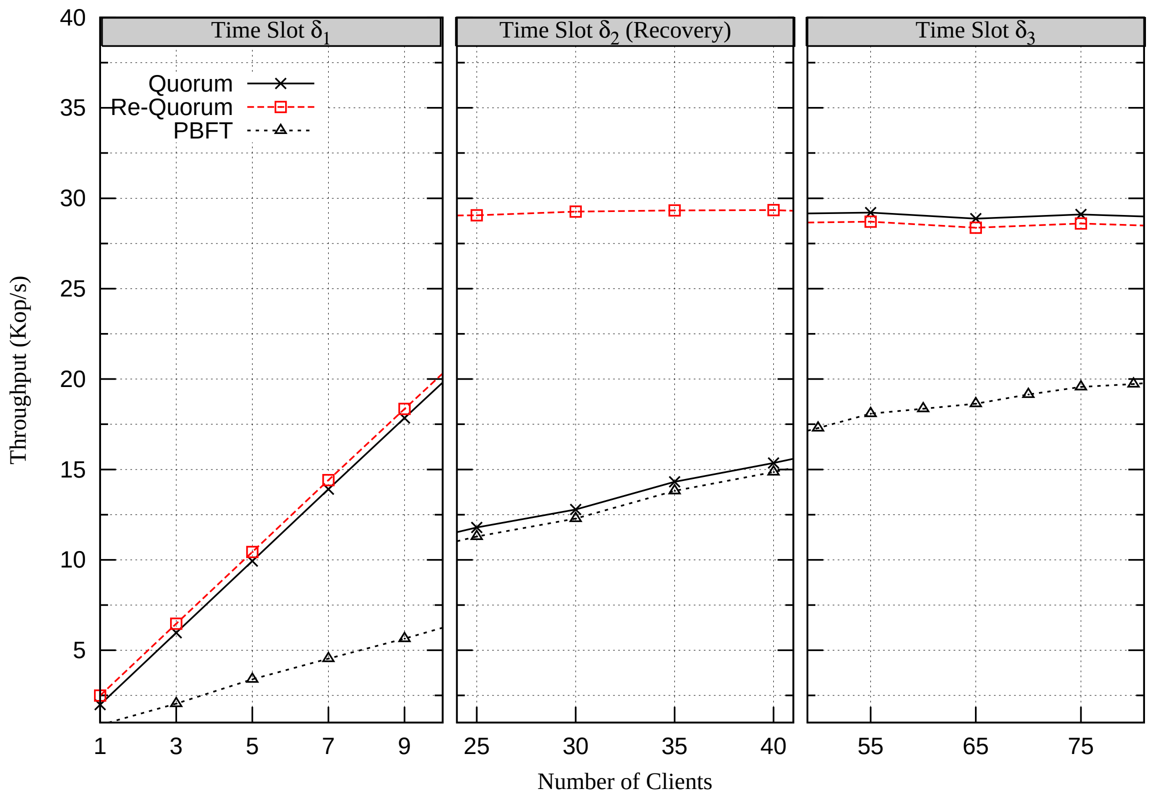


Figure 5.6: Throughput of Re-Quorum during three time slots.

Since the switching time (discussed above) does not exceed 50ms, i.e., the worst case, then the total switching time of Re-Abstracts is also 50ms (since Re-Abstracts switches only once, in contrast to Abstracts like Zlight, Quorum that need to switch twice while entering and leaving recovery). Assume that the time needed to switch from Zlight to

PBFT is negligible (in fact this is not true, but it has no great impact on the comparison). Consider, the average throughput of all protocols during each time slot as shown in Table 5.2. Now, if $\delta_1 = \delta_2 = \delta_3 = 1$ *hour*, then during $\delta = \delta_1 + \delta_2 + \delta_3$ the maximum throughput of PBFT will be $3600 \times (3K + 13K + 18K) = 3600 \times 34K$ operations. Whereas, Zlight completes $3600 \times (7K + 13K + 23K) = 3600 \times 43K$ since during the second time slot it recovers to PBFT-like protocol. Considering the switching time of Re-Zlight we get $3600 \times (7K + 24K + 23K) - (0.05 \times 23.5) = 3600 \times 54K - 720$. Thus, during the three-hours duration, the throughput of Re-Zlight exceeds Zlight by around $3600 \times 11K$ operations, i.e., around 20%. On the other hand, the overall throughput of Re-Zlight is around 37% better than that of PBFT.

Notice that the switching is always justified as long as the recovery time slot δ_2 is reasonably not small. In case the failure of some replica is transient (e.g., lost message, or flooded network, etc), then the replica does not need any maintenance or external intervention to repair, thus δ_2 will be small, and consequently it is worth nothing to switch to Re-Zlight. We think that if δ_2 is larger than one second then it is strictly fruitful for Re-Zlight to switch.

It is important to mention that our chosen time slots are just an example. Since observing Figure 5.5 conveys that any time slots can give almost the same results since the throughput discrepancy between Zlight and PBFT is large. This justification also applies for Figures 5.6 and 5.7.

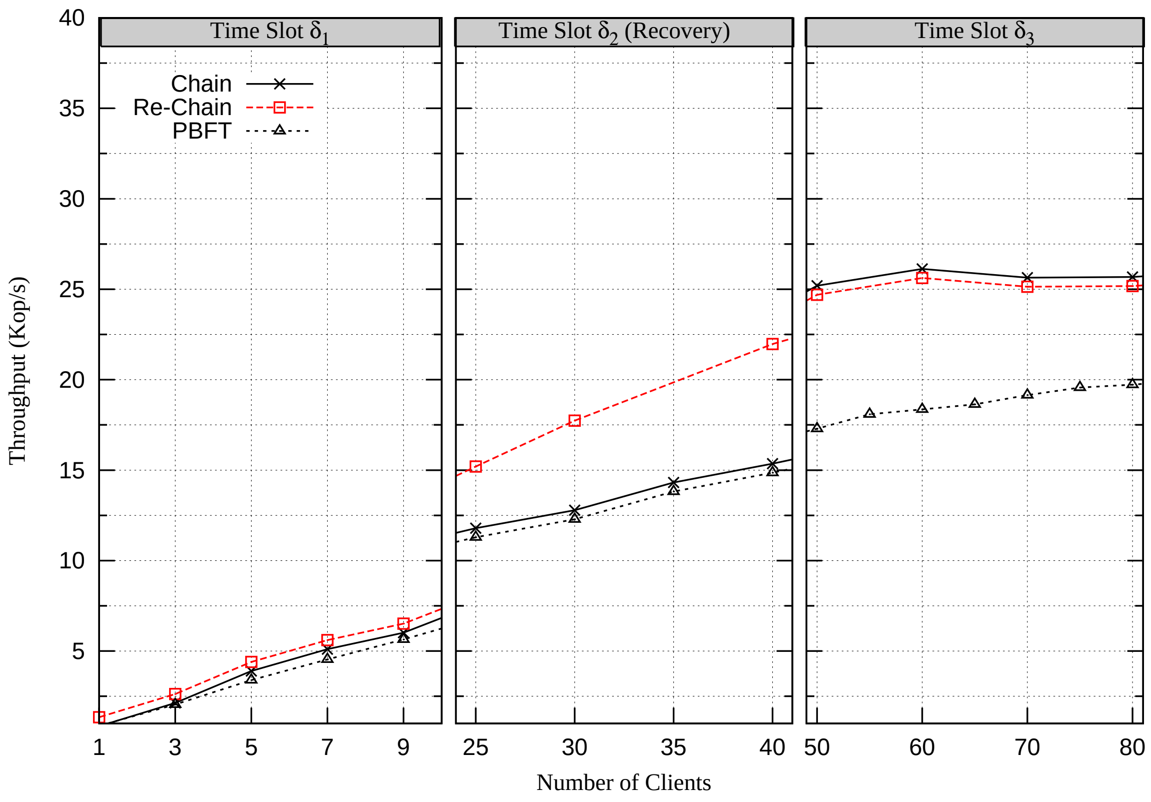


Figure 5.7: Throughput of Re-Chain during three time slots.

Re-Quorum. As for Re-Quorum, Figure 5.6 depicts similar results to Re-Zlight. In fact, since the difference in throughput between PBFT and Quorum is large, then in a short period of time Re-Quorum will outperform both Quorum and PBFT. The graph shows that in the three time slots Re-Quorum outperforms the PBFT by almost double throughput. This is referred to the minimal MAC authentications needed in Quorum and due to its short messaging pattern (see Figure 5.3). The overall gain in throughput of Re-Quorum over PBFT and Quorum during this duration is 74% and 23%, respectively. The calculated numbers are mentioned in Table 5.2.

Re-Chain. Similarly, Re-Chain outperforms Chain and PBFT in the three time slots as mentioned in Figure 5.7. However this time, the difference in throughput is less than that presented in the previous cases of Zlight and Quorum; especially for a small number of clients. This is expected since the chain-fashion message exchange pattern of Chain imposes large delays which impacts its throughput (Figure 5.3). However, Re-Chain achieves 10% of operations more than Chain during the three time slots in our scenario. Notice that according to the same graph, if the recovery occured in th first time slot, switching is not worthy as Re-Chain and PBFT are almost equivalent in throughput.

In the above graphs, we only considered messages of 0KB payloads. We don't reveal other payloads since they are close to the above results as the difference among protocols remains large even with large payloads. Instead, we convey a more interesting study where switching is not worthy.

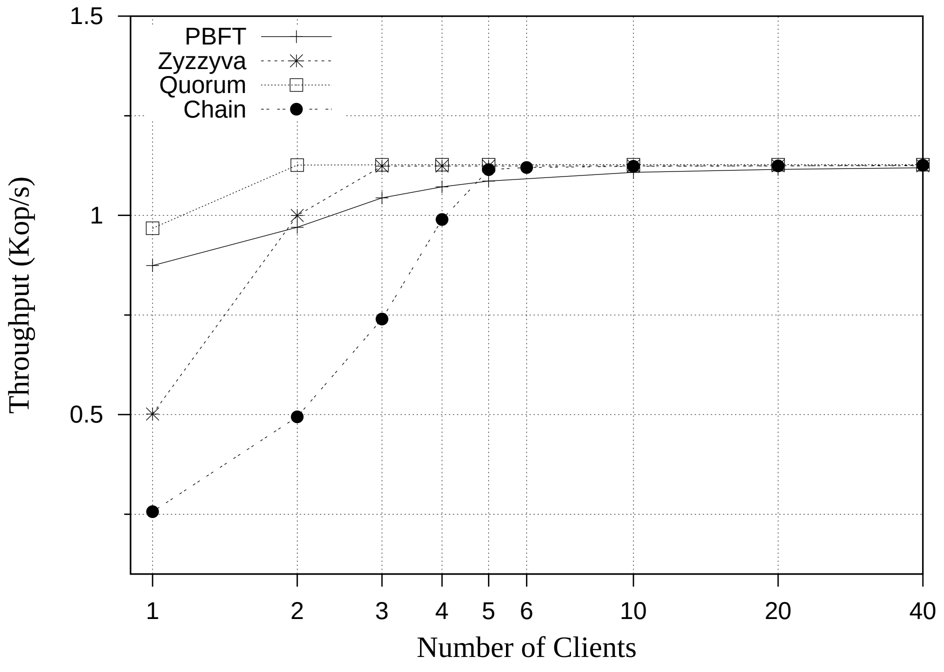


Figure 5.8: Throughput is negligible between BFT protocols when execution time is large.

Switching is not worthy. The above graphs are real experiments conducted using Redis as an application, knowing that Redis has small operation execution times. This made the difference in throughput among protocols significant. However, upon using OpenLDAP, that has operations of large execution time, the difference in throughput among different protocols becomes negligible especially with a large number of clients. This justification was elaborated in details in Chapter 3. Figure 5.8 depicts the throughput of different BFT protocols as a function of the number of clients. The graph shows that up to 5 clients, distinct protocols achieve different throughput. Quorum and Chain achieve the highest and lowest throughput, respectively. However, starting from 6 clients, the throughput of the protocols becomes roughly similar due to the CPU bottleneck. In this case, it is clear that switching is not worthy to occur, instead using PBFT can be more convenient since it can operate in the presence of failures.

5.4 Re-Abstract in the STATIC Mode

Our approach in this chapter presents a method that can be used to use any existing protocol as a member of Re-Abstract family. This conserves the performance of the protocol even under recovery. This approach is important for systems that does not exhibit a large variation in the underlying conditions. In this case, we have shown that such situation fits the STATIC mode, explained in the previous chapter, where the user needs to choose a protocol only once, i.e., at the installation of the service. Re-Abstract family guarantees that the user will have a constant (or an expected) performance even under recovery. If the system is quite fluctuating, however, the user can use the DYNAMIC mode for which we present a solution in the next chapter.

5.5 Lightweight Checkpointing

Since Re-Abstract family members are often speculative, then if the *abort history* AH is very large, switching overhead will be high; consequently, swtiching might not be worthy anymore. Lightweight checkpointing is used to maintain small size local histories LH on replicas in order to reduce the communication cost; in particular, upon moving from the Speculative phase to the Recovery phase. Lightweight checkpointing can minimize the local history sizes (256 for instance); and thus reduces the switching time.

Checkpointing appears crucial as Passive replicas have to be updated continuously, so that, they maintain an up-to-date state as Active replicas.

The pseudo-code of lightweight checkpointing is presented in Algorithm 0 and Algorithm 1. The client triggers checkpointing every k requests (we use k=256). It sends R_{CHK} message to the Active replicas (Algorithm 0, lines: 1 to 3) and starts a timer $Timer_1$. Each Active replica validates the R_{CHK} (MAC and times-tamp), and sends a CHK message to the client. A CHK_i message sent by replica i contains: the digest of 256 requests starting from its last checkpoint of number n_i, and the new supposed checkpoint counter n_i+1 (Algorithm 0, lines: 8 and 11).

When the client receives $2f$+1 CHK messages with valid MACs and matching digests, it sends a R_{CHK} message to the Passive replicas with the received LH piggybacked.

Otherwise, if $Timer_1$ expired, the checkpoint is postponed to the next attempt. After validating the request, a Passive replica appends LH to its local history (Algorithm 1, line 10), and replies back to the client with a CHK message. When the client receives f such messages from the Passive replicas, it sends a COM message to all replicas (Active and Passive) to commit the checkpoint (Algorithm 0, lines: 31 and 32). The replicas, then, truncate the 256 requests from their LH and increment their checkpoint counter n_i

(Algorithm 1, lines: 15). If the client is malicious and sent different checkpoint counter to different replicas, this will be discovered by replicas in in the next checkpoint phase, and the data will be recovered (this time with more than 256 requests).

A checkpoint does not succeed until all the replicas respond correctly. Thus, whenever a checkpoint fails, it is postponed, and the replicas truncate 2 × 256 requests instead of 256 in the next attempt. More generally, $i \times 256$ requests are truncated if i consecutive failed attempts occurred.

Algorithm 1 $CHK_{client}()$

```
1:                                                       ▷ Send $R_{CHK}$ to all Active replicas
2:  m ← $\langle R_{CHK} \rangle_{\mu_{c,i}}$
3:  Send(m,i) ∀ i ∈ Active
4:  $R_1 \leftarrow \phi$
5:  if [ thenUntil $Timer_1$ expires]$Timer_2() \neq \phi$
6:      while Receive($r_i$,i) ∧ vMAC($r_i$) ∧ i ∈ Active do
7:          if [ thencollecting 2f+1 matching CHK]$\|R_1\| \geq 2f+1$
8:              Break
9:          end if
10:         if $r_i$.LH=j.LH and $r_i.n_i = r_j.n_j$; $\forall j \in R_1$ then
11:             $R_1 \leftarrow R_1 \cup \{r_i\}$
12:         end if
13:     end while
14: end if
15:                                                      ▷ Send $R_{CHK}$ to all Passive replicas
16: $LH \leftarrow r_1.LH$
17: $m \leftarrow \langle R_{CHK}, D(LH) \rangle_{\mu_{c,i}}$
18: Send(m,i) ∀ i ∈ Passive
19: $R_2 \leftarrow \phi$
20: if [ thenUntil $Timer_2$ expires]$Timer_2() \neq \phi$
21:     while Receive($r_i$,i) ∧ vMAC($r_i$) ∧ i ∈ Passive do
22:         if [ thencollecting f+1 matching CHK]$\|R_2\| \geq f+1$
23:             Break
24:         end if
25:         if $r_i.LH = r_j.LH$ and $r_i.n_i = r_j.n_j$; $\forall j \in R_1 \cup R_2$ then
26:             $R_2 \leftarrow R_2 \cup \{r_i\}$
27:         end if
28:     end while
29: end if
30:                                                      ▷ Commit checkpoint $n_c$
31: if TRUE then
32:     $m \leftarrow \langle COM, n_i \rangle_{\mu_{c,i}}$
33:     Send(m,i) ∀ i ∈ Σ
34: end if
```

Algorithm 2 $CHK_{replica}(m)$

```
1:                                                       ▷ Handling checkpoint requests
2:  if MAC(m) is False or $m.t_c \leq t_i[c]$ then return False
3:                                                       ▷ request is not valid
4:  end if
5:  if m.type=$R_{CHK}$ then
6:      if i ∈ Active then
7:          $r \leftarrow \langle CHK, D(LH_i), n_i + 1 \rangle_{\mu_{c,i}}$
8:      else if [ thenPassive replicas update their LH]
9:          $LH_i \leftarrow m.LH$
10:         $r \leftarrow \langle CHK, D(LH_i), n_i + 1 \rangle_{\mu_{c,i}}$
11:     end if
12:     Send(r,c)
13: else if [ thenCommit checkpoint]m.type=COM
14:     $n_i \leftarrow n_i + 1$
15:     Truncate($LH_i, n_i$) truncate 256 LH entries
16: end ifreturn True
```

5.6 Conclusion

In this chapter, we presented *Re-Abstract* family that allows an existing BFT protocol to recover to itself. This helps in maintaining the performance of the same protocol during recovery. This approach is very useful in the STATIC mode of the selection model presented in the previous chapter. The mechanism is convenient to systems that do not exhibit too fluttering conditions.

Our approach introduces a cheap recovery phase that eliminates the faulty or slow replicas and replaces them with correct ones. Re-Abstracts require at least $4f+1$ replicas to maintain one-copy semantics if up to f replicas are faulty. We introduced the design of the recovery phase using Re-Zlight as an example; and we provided solutions for availability and contention.

We conducted some experiments on two real application Redis and OpenLDAP. On Redis, we have shown that, using our approach, existing protocols like Zlight (an abortable version of Zyzzyva), Quorum, and Chain achieve from 20% to 100% better throughput during recovery. The experiements demonstrated that switching in Re-Abstract approach is worthy when the recovery duration is long (i.e., it takes time to repair a faulty node). However, if the recovery time is short (e.g., ;less than 1 second) or the executuion time of the used application is large (e.g., OpenLDAP), then switching will not render a significant profit.

Chapter 6

Can BFT be Adapt-able?

In this chapter, we introduce a novel BFT system called Adapt. The system uses existing, and probably future, BFT protocols together, at once, and adapts to the underlying system through switching dynamically from one protocol to the other according to a dynamic switching policy. The switching policy follows the Dynamic and Heursitic modes of the selection mechanism introduced in Chapter 4. A prediction function, based on Support Vector Machines, predicts the performance of the protocols in real-time to choose the preferred protocol that fits the next stage, i.e., if the underlying conditions of the system change frequently. The evaluation of this approach investigates when switching is worthy. In addition, it conveys the high accuracy of prediction and computes the overall gain in performance.

6.1 Why Adapt?

Various BFT protocols have been developed recently to fill the gaps of their predecessors; however, as demonstrated in Chapter 3, *no BFT protocol fits all*. *Abortable BFT* (presented in Chapter 2) introduced the possibility to collect state-of-the-art protocols in one package and *switch* between them when *something wrong happens*; nevertheless, without introducing a clear *switching policy* this approach remained limited.

Furthermore, the previous chapter provided a method to have any Abortable BFT protocol recover to itself in order to persist on its anticipated performance. However, this mechanism addresses systems that do not experience frequent changes in the underlying conditions; in fact, it is mostly designed to stable systems with occasional failures. This approach is convenient to work in the Static mode of the selection model presented in Chapter 4.

Here we introduce an example where the static mode, and consequently Re-Abstract approach, are sometimes insufficient. Figures 6.1 and 6.2 explore the throughput of well-known BFT protocols using 0/0 and 4/0 micro-benchmarks (used in [15]), respectively. The results show that no protocol outperforms the others as the number of clients changes. Figure 6.1 demonstrates that Quorum [26] performs 50% better than the nearest competitor Zlight [1] [26] when the number of clients is small. As this number increases, Quorum crashes due to high contention, and Zlight becomes the leader. Another look at Figure 6.2 shows that all the protocols dominate Ring [1] with few clients. However, when the number of clients exceeds 200, the network becomes a bottleneck, and hence Ring outperforms

[1]Quorum is a $3f+1$ abortable version of Q/U [2], and Zlight is an abortable version of Zyzzyva [32].

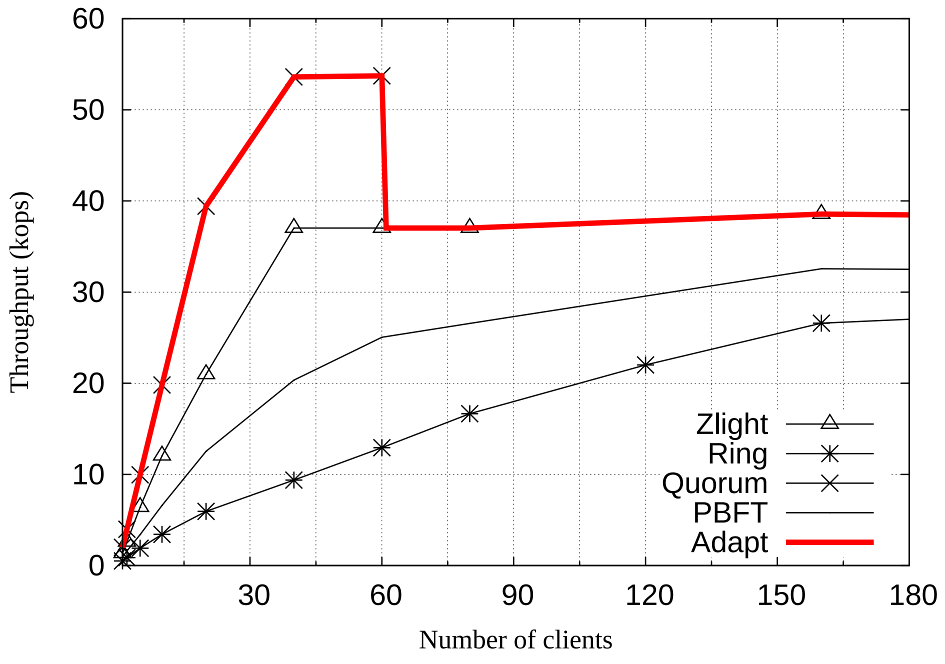


Figure 6.1: Throughput of state-of-the-art BFT protocols using the 0/0 micro-benchmark.

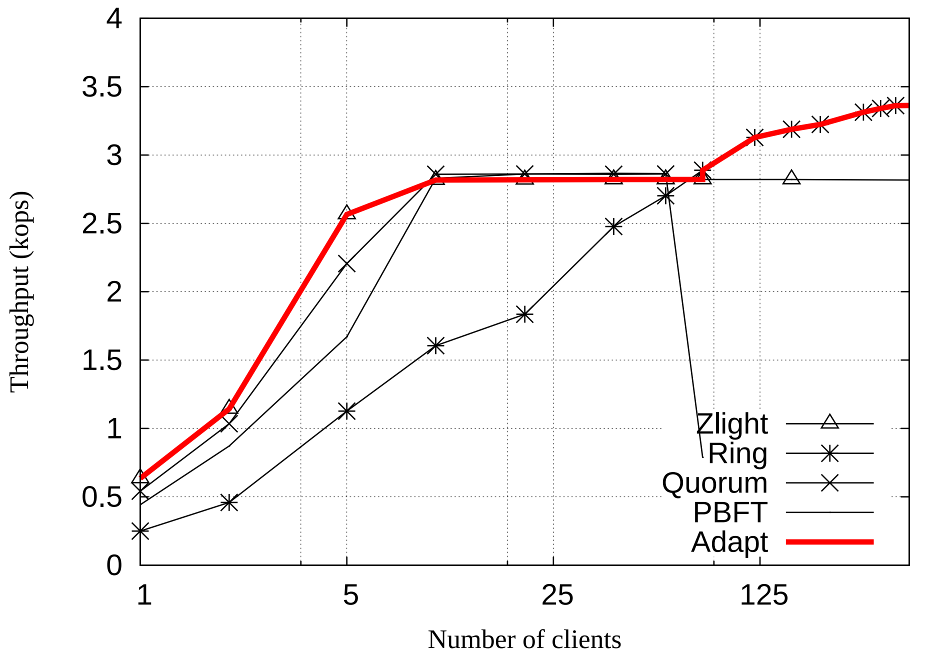


Figure 6.2: Throughput of state-of-the-art BFT protocols using the 4/0 micro-benchmark (log. scale).

the other protocols by 25%. Therefore, upon using Re-Abstrct approach, a single protocol will be used at once, and thus recovery to itself might not be fruitful. Instead, it is very interesting if one can choose the optimal use of all protocols in any condition.

In this chapter, we present Adapt; a BFT system to make BFT protocols adapt to the system state changes, i.e., to the underlying system conditions like contention, message size, processing time, etc. Adapt is designed to make an optimal use of the existing

BFT protocols. At anytime, a running protocol can be aborted and replaced by another protocol according to some change (an *event*) in the underlying system conditions. The launched protocol is chosen after passing a real-time evaluation process by using the Dynamic and the Heuristic modes of the selection machanism introduced in Chapter 4. Adapt takes into consideration both: robustness and performance. We use *Support Vector Machines* prediction mechanisms to assess performance in real-time. In addition, we discuss when it is worthy to switch from one protocol to another. This approach ensures that the optimal choice among the existing protocols is always chosen to run in the next phase. Adapt uses some heuristics (in the Heuristic mode) to optimize the switching process.

To explore the power of Adapt, we give an example using some numbers from the experiments we conducted (presented also in Figures 6.1 and 6.2). Consider a system running for a three-hours duration: one hour with no contention (less than 20 clients), one hour with tens of clients (more than 40), and one hour where the network is jammed with big requests (4KB) issued by more than 300 clients. In this scenario, one might choose Zlight as the most convenient protocol according to Figures 6.1 and 6.2 (the curve with triangle line-points). During this period, Zlight can handle a maximum of: $(21kops+38kops+2.8kops)\times3600=223$ Mops. On the other hand, Adapt can launch Quorum in the first phase, Zlight in the second, and Ring in the last. Adapt handles a maximum of: $(40kops+38kops+3.4kops)\times3600=293$ Mops. The graph of Adapt is depicted in red color on Figures 6.1 and 6.2. Thus, Adapt handles 70 extra million operations (i.e., 30% gain) during this period. Additional cost should be payed for switching between phases, but this is negligible as shown in Chapter 5. Notice that, in case a replica failure is detected, Adapt launches PBFT, while the other protocols either crash or go into a very expensive recovery phase.

6.2 System Overview

Adapt is composed of three systems: BFT Subsystem (BFTS), Event Subsystem (ES), and Intelligent Control Subsystem (QCS). BFTS is a pool of BFT protocols, it represents the BFT and the *abortability* libraries. ES monitors the system, and detects the changes in the *Impact Factors* (e.g., number of clients, request size, data loss, etc) of the underlying system, if any. Upon detecting a *significant* change in the impact factors, ES sends an *event* to the *QCS*, triggering an evaluation phase. QCS maintains an optimal system performance by running the preferred protocol under the current conditions. QCS evaluates the BFT protocols by calculating their *Evaluation Score* (E); the protocol with the higher E is launched in the next phase. The evaluation phase works according to the Dynamic and Heuristic modes of the selection model and selection algorithm in Chapter 4. QCS predicts the KPI values of each protocol, in real-time, through using SVR [21]. *Switching* to a new protocol takes place only if it is worthy. This chapter is concerned mainly with QCS; whereas, ES will appear in future work.

6.3 SVM Prediction Mechanisms

Using analytical models like [48, 45] can give a general idea about the real performance of BFT protocols; however, these models could not give accurate predictions for a detailed *fine-grained* performance as the system conditions vary. This refers to the various parameters that can affect the behavior of BFT protocols like: network properties, network topology, platform type, operating systems, etc. This was discussed thoroughly in Chapter 3. On the contrary, prediction mechanisms that depend on the observations of actual data, derived from the real system during testing for example, can be more accurate. *Machine learning* predictors like *SVR* [21] are often used for this goal. Adapt learns about the BFT protocols by training a prediction function on a real data collected throughout the testing phase of system deployment, and keeps learning as long as the system is operational. The prediction function is used then to predict the real-time performance of the protocols, mainly the KPIs.

In this section, we briefly explore the SVR mechanism and then we mention the evaluation methods that we use to evaluate prediction accuracy.

6.3.1 Methodology of Support Vector Regression (SVR)

This section introduces SVR [21], [49]. We use SVR to predict the throughput of BFT protocols as the system conditions change, e.g., number of clients, request size, etc.

Let x be an instance of some request transfer. An instance is represented by a d-dimensional feature vector x $=(x^{(1)},...,x^{(d)})^T \in \Re^d$. Each dimension is called a feature (e.g., number of clients, request size, etc). The target $y \in R$ of an instance x is the predicted value (e.g., throughput, latency, etc). In this paper, we refer to features and targets as *Impact Factors* and *Key Performance Indicator* (KPIs), respectively. Our goal is to train a regression function $f : \Re^d \rightarrow \Re$ that maps any instance x to its target y (e.g., maps the number of clients to throughput).

A training set of n previously known instance-target pairs $(x_1, y_1), ..., (x_n, y_n)$ is required to train the regression function. A family of candidate regression functions F must also be specified, e.g., the set of all linear functions. A loss function $L(f(x), y)$ is used to measure how close a prediction $f(x)$ is to the actual target y. We use the ϵ-insensitive loss $L(f(x), y) = max(|f(x) - y| - \epsilon, 0)$. This loss function is standard in support vector regression [49]. Training a regression function involves finding a function $f \in F$ that minimizes the loss on the training set, plus a regularization term as follows:

$$\hat{f} = arg\ \underset{f \in F}{min} C \sum_{i=1}^{n} L(f(x_i), y_i) + \|f\|^2 . \tag{6.1}$$

The first term is known as the empirical loss. The prediction function is good if it minimizes the empirical loss on the training set, however, it may not be the one that produces the most accurate predictions on future test instances. This is because minimizing the empirical loss has the danger of over-fitting the training data (since it is a small sample out of a big population). One way to prevent over-fitting is to regularize f by its norm $\|f\|^2$, with the intuition that we prefer a smoother regression function. The scalar C balances the empirical loss and the smoothness of f. The solution to the optimization problem 6.1 can be found efficiently using a quadratic program.

We use the *Radial Basis Function* (RBF) kernel, which induces a rich candidate regression functions family. The kernel enables learning of highly non-linear, yet well regularized, regression functions. We use the SVM library: *LIBSVM* [16].

6.3.2 Prediction Model Selection

Model selection is performed to choose *good* values for the prediction function parameters (e.g., C and γ in the RBF Kernel) in such a way to make a trade-off between prediction model accuracy and complexity. This is basic since choosing parameters that yields high accuracy may cause over-fitting the training data, and consequently predicting unknown data will not be accurate. Grid-search [30] and v-fold cross-validation [13] are often used for model selection. Various pairs of (C,γ) values are tried using grid-search, and the one with the best cross-validation accuracy is picked.

Grid-search is a naive, yet powerful, method for model selection. It probes a defined domain $[a, b]$ with a step δ. Starting from $x = a$ and until reaching b, the model is cross-validated on x; then x is incremented by δ, and so until reaching the *best* values of C and γ.

Cross-validation is used along with grid-search (for each value x). The idea of v-fold cross-validation is to split the original training set into v equal subsets; the prediction function is trained on $v - 1$ subsets and tested on the remaining one. This is repeated v folds until each instance of the whole training set is predicted once. Cross-validation accuracy is the percentage of data which are correctly predicted with a predefined error.

6.3.3 Evaluating Prediction Accuracy

We use the relative prediction error E, defined in [29], to evaluate the accuracy of an individual target (e.g., throughput) prediction. Denoting the actual target by R and the predicted target by $\hat{R}$, we define the relative prediction error as follows:

$$E = \frac{\hat{R} - R}{R} \tag{6.2}$$

To evaluate the overall predictions of the target on a protocol, we use the *average relative prediction error*:

$$A = \frac{1}{n}\sum_{i=1}^{n}\frac{|\hat{R} - R|}{R} \times 100 \tag{6.3}$$

6.4 Adapt Architecture

Adapt is composed of three subsystems: BFT Subsystem (BFTS), Event Subsystem (ES), and Quality Control Subsystem (QCS). The term *eState* represents the state of the entire system: the infrastructure, the platform, the exchanged messages, the installed software (Adapt and other applications), etc.

6.4.1 BFT Subsystem: BFTS

BFTS is the BFT library. It is composed of a collection of state-of-the-art protocols like PBFT [15], Zyzzyva [32], Quorum [26], Ring [1], Chain [26], etc. To remove confusion, sometimes we call them sub-protocols: *SP*. These SPs are designed to be *abortable* [26]. A running *SP* sp_i can be interrupted at any time, finishing the current *Phase*, and starting a new Phase by launching another SP sp_j; we call this process: *switching*. Switching conserves the local states of the replicas across different *Phases* by collecting the *abort history* AH of the operations executed in the current Phase, and initializing the replicas (i.e., executing AH) in the next Phase. Our switching concepts are derived from [27] and [26] as shown in Chapter 2, thus we do not explain switching further in this chapter.

6.4.2 Event Subsystem: ES

ES monitors the system and collects the *meta-data* that construct the eState (i.e., network state, messages exchanged, clients statistics, etc). These meta-data are the *Impact Factors* that can affect the performance and reliability of the SPs. ES periodically sends event notifications to QCS, updating its database, and informing it with the new eState. ES is a plan for future work, thus it is out of the scope of this thesis.

6.4.3 Quality Control Subsystem: QCS

QCS is the control unit of Adapt; it is in charge of taking the decisions: whether switching is needed due to some changes in eState, and which SP is the preferred to launch in the next Phase. QCS evaluates the SPs by computing the *Evaluation Scores (E)* described in Chapter 4. Some of the parameters of E, more specifically the KPIs, are predicted using SVR [21]. Evaluation in QCS is triggered upon receiving an event from ES; whereas, switching takes place only if it is worthy (more details next).

This chapter focuses on QCS and the accuracy of predictions.

6.5 Quality Control Subsystem (QCS)

The role of the Quality Control Subsystem (QCS) is axial in Adapt. Adapt performs two important tasks: (1) Performing run time evaluations for protocols according to the dynamic and heuristic modes of the selection model introduced in Chapter 4 and (2) predict the β-KPIs used by this model, also at run time, to make the switching process among protocols dynamic. This yields a powerful system that achieves optimal robustness and performance through joining the merits of existing protocols.

In this section, we start by exploring the prediction mechanism used in QCS, and then we move into discussing the evaluation of protocols and switching.

6.5.1 Prediction of β-KPIs

Before diving into details, we recall the matrix of β-KPIs already used in the selection model in Chapter 4:

$$B = \begin{pmatrix} \beta_{11} & \beta_{12} & \beta_{1b} \\ \beta_{21} & \beta_{22} & \beta_{2b} \\ \vdots & \vdots & \vdots \\ \beta_{n1} & \beta_{n2} & \beta_{nb} \end{pmatrix}$$

In this section we describe the method used to predict the entires of the above matrix using SVR.

6.5.1.1 System eState

The β-KPIs must always be known to compute the evaluation scores (E) of the protocols. β-KPIs vary as eState changes. We define eState by $S = \{s_i = (f_1, f_2, ..., f_j, ..., f_m)\}$; where m is the number of Impact Factors. The values f_j represent the impact factors of the system state. Various impact factors (f_j) can affect the performance of the protocols in BFTS; the following represents a classification for some impact factors:

- Network factors: network speed, packet loss, packet duplication, number of clients, etc.
- Data factors: request size, response size, batching, etc.
- Application factors: execution time, memory needed, etc.
- Infrastructure factors: CPU speed, memory size, virtualization, etc.

QCS runs the prediction process to assess the values of β-KPIs for each eState.

6.5.1.2 Prediction Mechanism

To predict the run time values of β-KPIs, we use *SVR* with the *Radial Basis Function* (RBF) Kernel (see [11] and [49]). We define a prediction function $f_{C,\gamma} : \mathbb{S} \longmapsto \Re$, where $\mathbb{S}$ is the set of eStates defined above. C and γ are the free parameters chosen through training the function (see later). Training takes place on an initial data set called: *Training Set*. After choosing the parameters C and γ; for any SP sp_i, instant t, and eState $s_t = (f_1, f_2, ..., f_j, ..., f_m)$, the prediction function can be used to predict the value of β_{i,s_t} as follows:

$$\hat{\beta}_{i,s_t} = f_{C,\gamma}(s_t). \tag{6.4}$$

The prediction of a β-KPI takes place in two main steps:

- Training: Train the prediction function on a *Training Set* for model selection.
- Prediction: Run the prediction function to assess the value of β, given a new eState.

Training. QCS trains the prediction function using the Training Set. The Training Set of a protocol is composed of instances of eState tuples $s_t = (f_{1,t}, f_{2,t}, ..., f_{m,t}) \in \mathbb{S}$ for an instant t, associated with the corresponding β-KPI values [2]. The structure of the training set is as follows:

$$\left\{\begin{array}{c} s_1 = (f_{1,1}, f_{2,1}, ..., f_{m,1}) \\ s_2 = (f_{1,2}, f_{2,2}, ..., f_{m,2}) \\ ... \\ ... \\ s_l = (f_{1,l}, f_{2,l}, ..., f_{m,l}) \end{array}\right\} \longrightarrow \left\{\begin{array}{c} \beta_{i,1,k} \\ \beta_{i,2,k} \\ ... \\ ... \\ \beta_{i,l,k} \end{array}\right\} \tag{6.5}$$

[2] In SVM terminology, f and β-KPI correspond to features and target, respectively.

The left part of the Training Set 6.5 lists l different eState tuples composed of m impact factors (e.g., number of clients, request size, etc.). The right hand side shows the corresponding values of some $\beta_{i,k}$ (e.g., throughput) of an SP sp_i.

Different eState tuples are obtained as the impact factors vary. An initial training set is collected before starting the system (e.g., while testing). Afterwards, upon normal system operation, additional instances can be appended to the training set, trying to widen the knowledge base of the system further.

Model Selection. A prediction model selection must be performed before using the prediction function. *Grid-search* [30] and v-fold cross-validation [13] are used to identify the parameters C and γ of Equation 6.4. The method is explained further in Subsection 6.3.

Prediction. When the parameters of the prediction function are identified, the function becomes ready for prediction. At any time t, the system can predict the value of a β-KPI for an eState $s_t = (f_{1,t}, f_{2,t}, ..., f_{m,t})$. Prediction takes place by solving Equation 6.4. To measure prediction accuracy, the prediction function is tested on a *Test set* to get a predicted value of β: $\hat{\beta}$. The Test set is similar in structure to the Training set but it comprises new data instances (i.e., not used in training). The relative error between β and $\hat{\beta}$ is then calculated (see Subsection 6.3). We use LIBSVM [16] as an implementation of SVR.

6.6 Evaluating Prediction Accuracy

6.6.1 Experimental Settings

To initialize a Training Set, we conducted extensive experiments on some BFT protocols while changing some Impact Factors. The output is then parsed to match the input format of LIBSVM [16]. LIBSVM is used to train the prediction function ϵ-SVR using the *RBF* kernel [49]. In our experiments, we consider the protocols: PBFT, Zlight, Quorum, Chain, and Ring. Other protocols like Q/U [2] and HQ [22] are similar to Quorum and PBFT, hence we exclude them.

BFT experiments are performed on *Ubuntu* OS, installed on 25 64-*bit* $Xeon$ machines with 2 GB of memory, and deployed on Emulab [55] testbed. We use the advanced key-value store Redis [46] as an application. Redis server is installed on all system replicas. The number of replicas is four and the fault factor f is equal to one. Each replica runs on a separate machine and the client processes share 20 machines. All nodes communicate through a Fast Ethernet network.

To compare results, we use the standard *a/b* benchmark [3] with different payload sizes: 0/0, 4/0. This is the same benchmark used in [15].

6.6.2 Evaluation Parameters

To evaluate the prediction accuracy of Adapt, we try to predict the throughput of different protocols under different conditions. We consider the *Impact Factors*: number of clients and request size. These factors have the greatest influence on the performance of BFT protocols. We considered 9 request size values (in Bytes): 0, 64, 128, 256, 512, 768, 1024,

[3]In a/b benchmarks, a, and b correspond to request size, and response size in KB, respectively.

2048, 3064, and 4096. The number of simultaneous clients ranged from 1 to 400 clients. We run more frequent experiments as the number of clients decreases to have better data representation [4]; this is needed since the throughput of BFT protocols is more fluttering with few clients. We conducted around 400 experiment runs for each protocol.

Though we believe that these parameters can evaluate our approach correctly, we look forward towards more accurate evaluation. Therefore, our future plan is to use more KPI parameters like throughput, latency, capacity, etc. Similarly, we will increase the number of impact factors: number of clients, request size, response size, processing time, malicious behaviors, etc.

6.6.3 Choosing SVR Parameters

We divide the data instances (i.e, the results) of the experiments into: a Training Set, and a Test Set. The former is used to train the prediction function, and the Test Set is used to evaluate its accuracy. We get the Test Set by, randomly, selecting 15% (around 50/400) of the data instances; the remaining 85% form the Training Set.

To choose an optimal model parameters: C and γ, we used the *grid.py* [30] script, in LIBSVM library, and the *five-fold* cross-validation [13]. Table 6.1 represents the values of C and γ that we achieved in this study.

	PBFT	Zlight	Quorum	Ring	Chain
C	10^6	5×10^6	20×10^6	10^6	1.7×10^6
γ	5×10^{-4}	1.2×10^{-4}	0.1×10^{-4}	10^{-4}	6.5×10^{-5}

Table 6.1: SVR parameters (γ and C).

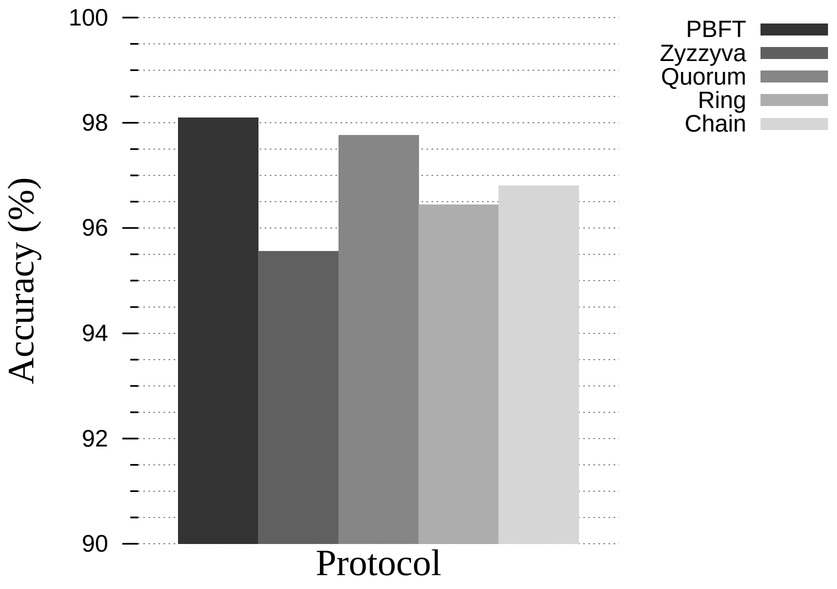


Figure 6.3: Average relative prediction accuracy.

[4]For instance, we ran experiments for all the values from 1 to 5 clients, then we widen the range gradually up to 300 clients, where the step between each two consecutive experiments becomes 50 clients.

6.6.4 Average Prediction Accuracy

This section explores the prediction accuracy we achieved for the protocols: PBFT, Zlight, Quorum, Chain, and Ring. Figure 6.3 presents the average prediction error of these SPs. The figure shows that the error ranges between 1.9% and 4.5% depending on the protocol. In the next sections we explain how do different protocols affect prediction accuracy. Since the accuracy, in all cases, is more than 95%, it is considered high enough to decide whether switching is worthy or not since the discrepency in performance among different protocols is significanlty higher than 5% in many cases. We explore how this affects the switching process given a *switching threshold*. This is crucial in our approach since switching is worthy only when the difference in performance among the protocols is larger than the switching threshold.

The achieved accuracy in Figure 6.3 can change in two cases. First, adding new parmeters to the function will decrease this accuracy. For instance, considering malicious clients, message dropping, etc., in the evaluation will have a negative impact on prediction accuracy. On the other hand, incresing the size of the Training set will increase accuracy. In fact, we argue that considering more parameters in the evaluation requires larger Training sets (which are not hard to achieve).

Observing the histogram in Figure 6.3, we notice that the prediction accuracy is not similar for the different protocols, though we are using the same SVR kernel function and the same level of cross-validation folds (to decide the parameters of the prediction function). In fact, this is referred to the behavior of the protocol itself as the underlying conditions change, e.g., request size, response size, and contention. We explain this further in the following sections.

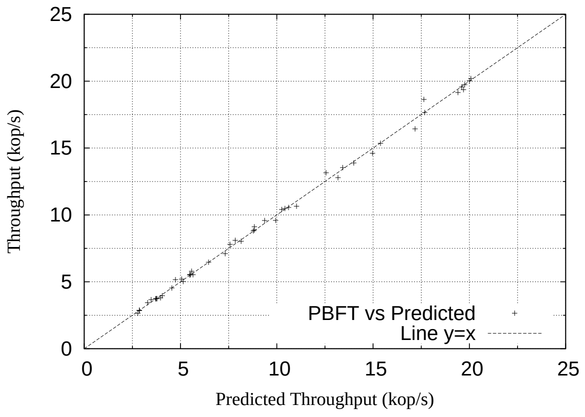
6.6.5 Scattered Diagrams

Figures 6.4 and 6.5 depict the scattered diagrams of the predicted points of the different protocols. The x-axis represents the predicted throughput while the y-axis represents the real throughput. The closer the point is to line $y{=}x$, the more accurate is the prediction (i.e., nearest to $y{=}x$ is better). These graphs give a wider view about the predicted points. Each graph plots around 50 predicted points out of 400 real ones. The diagrams clearly show how close are the points to the line $y = x$. Most of the points are coinciding with this line.

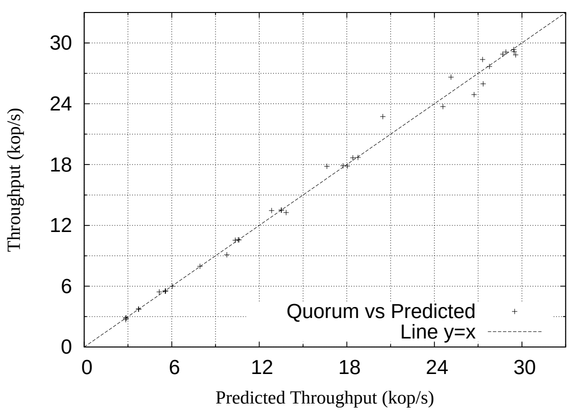
Notice that the accuracy we have seen in the previous section seems low with respect to these points. In fact, though most of the points are very close to real values (i.e., around 1%), the few points that are predicted with higher error are making this variation. This can be observed clearly in Subfigures 6.5(a) 6.5(b). We argue that with further training (more than 400 tries in our case) more accurate and stable predictions could be obtained. This is expected since this variation is a consequence of the unstability of the protocols in the experiments we have done. With more runs, an accurate training set can be gathered.

In Subfigure 6.5(b) the predicted points are a bit far away from the line $y = x$ more than in other protocols. Better results could be achieved for Zlight being a stable protocol, however, we noticed a lot of fluctuations in the actual throughput of Zlight in our experiments. This is unusual in Zlight, perhaps Emulab had routing issues at that period [5].

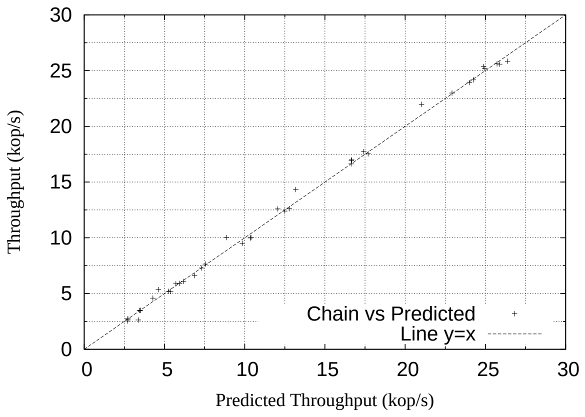
[5]Zlight uses Mrouted to multicast requests, thats why other protocols were not greatly affected.



(a) PBFT



(b) Quorum

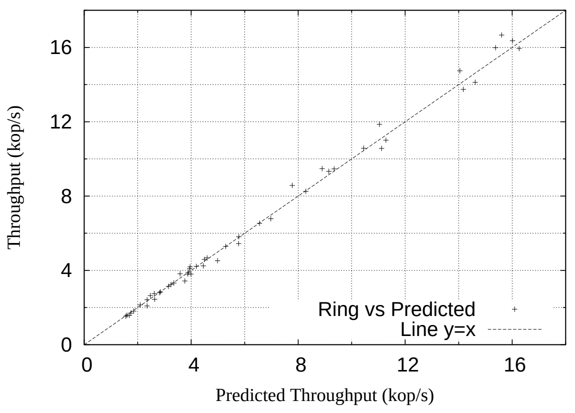


(c) Chain

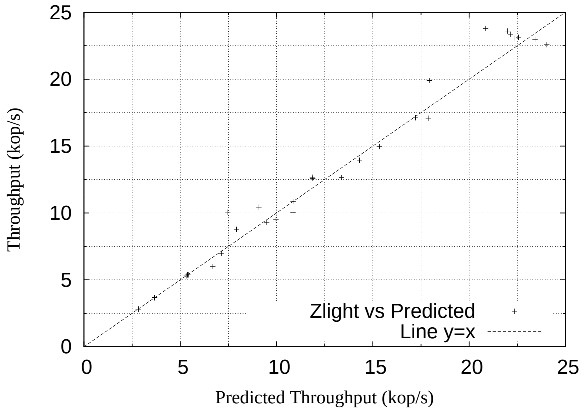
Figure 6.4: Predicted vs Real throughput of PBFT, Quorum, and Chain.

6.6.6 Levels of Prediction Error

The average prediction error gives a general view about accuracy; nevertheless, a closer look at Figures 6.6 and 6.7 convey how the prediction accuracy is changing. The x-axis of the graph represents the level of prediction error from 1% to 10%; the y-axis shows the percentage of commulative prediction errors. In such graphs, when the curve is monotonic with a big slope, then the predictions are quite changing (the error is quickly augmented)



(a) Ring



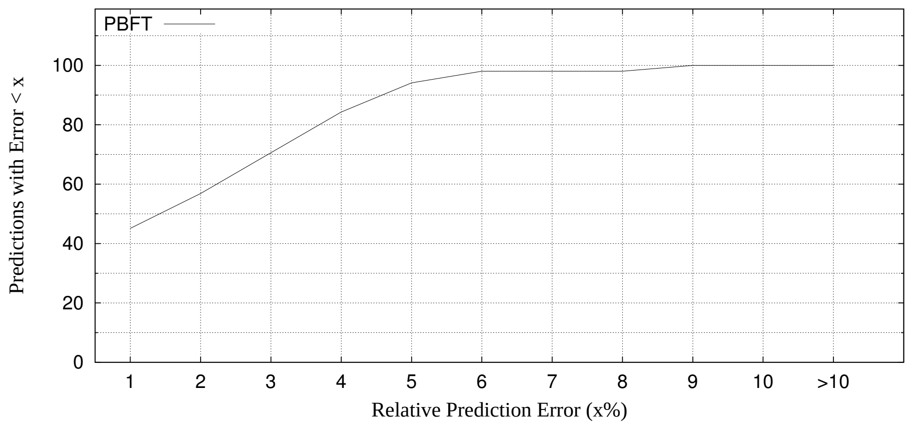
(b) Zlight

Figure 6.5: Predicted vs Real throughput of Ring and Zlight.

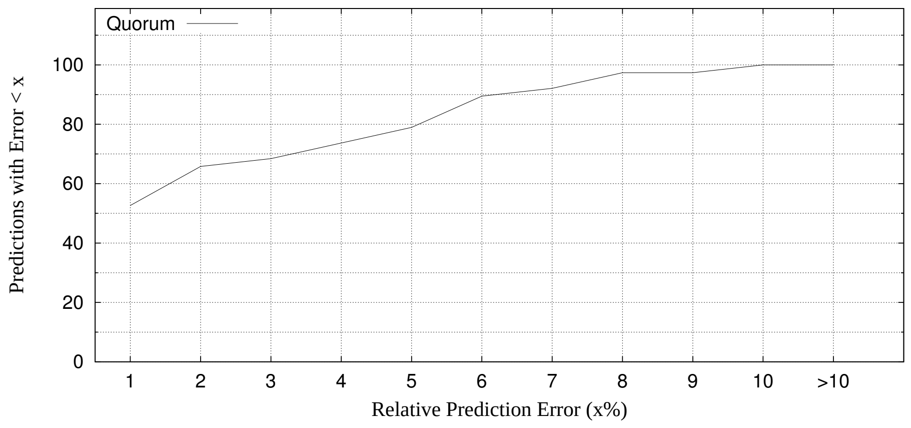
starting from 1% to 10%; otherwise, the curve is more like a plateau, which means no big prediction errors with this percentage level.

The graphs in Figures 6.6 and 6.7 convey that the prediction accuracy is higher than 99% in 40% to 50% of the cases (notice the start point of the curves). Ring is the exception, since the throughput of Ring increases slowly as the number of clients increases which makes the prediction difficult with such small training set. In fact, the throughput of Ring becomes steady with more than 400 clients accessing the system. For this same reason, the curve of Ring (in Subfigure 6.7(b)) has a big slope. A similar situation can be noticed with Quorum by observing Subfigure 6.6(b). The curve between 1% and 5% has a big slope, i.e., signifcant errors are found from 1% up to 5%. The reason behind this is also the quick change in the throughput of Quorum with few clients. With more than 5%, no prediction errors are almost faced as Quorum's throughput becomes steady and thus prediction becomes easier (notice that the curve becomes horizontal). Again regarding Zlight in Subfigure 6.7(a), the prediction errors with more than 5% are frequent (the curve has a large slope). This is due to the unfamiliar behavior of Zlight in the original experiments, i.e., the training set, hence we expected the prediction to skew.

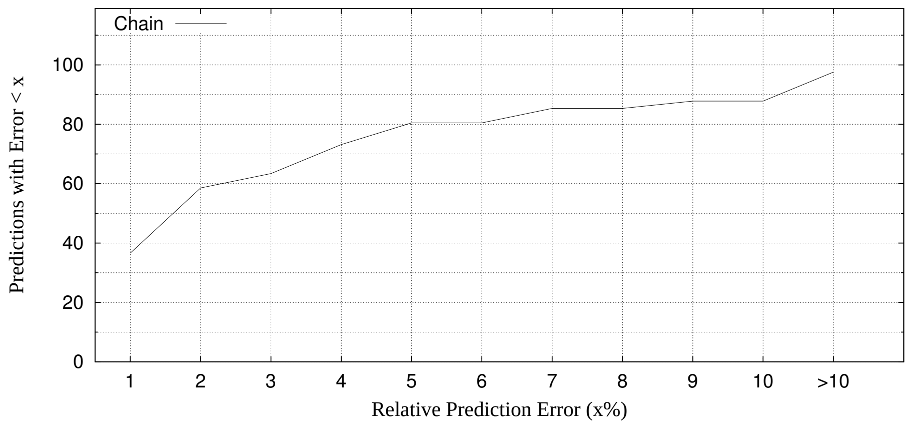
In addition, the Figures 6.6 and 6.7 indicate that the percentage of predictions degrades gradually as L increases since all curves advance like a plateau with a higher error value. We argue that it is hard to achieve better prediction results with such training set of



(a) PBFT



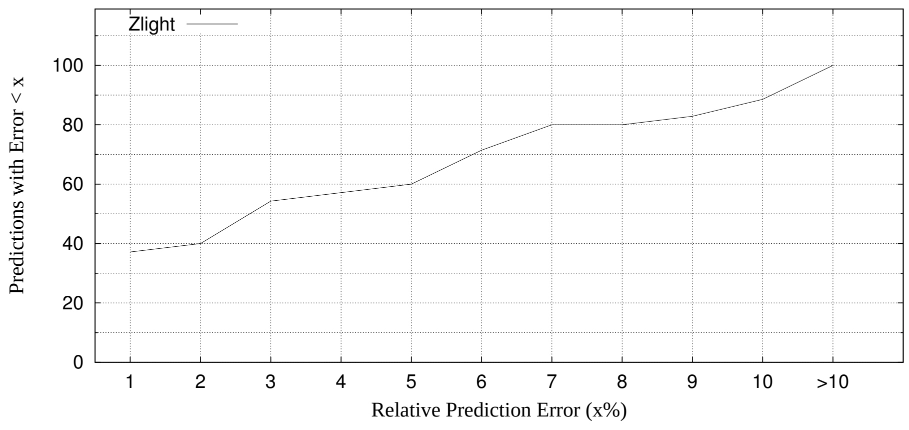
(b) Quorum



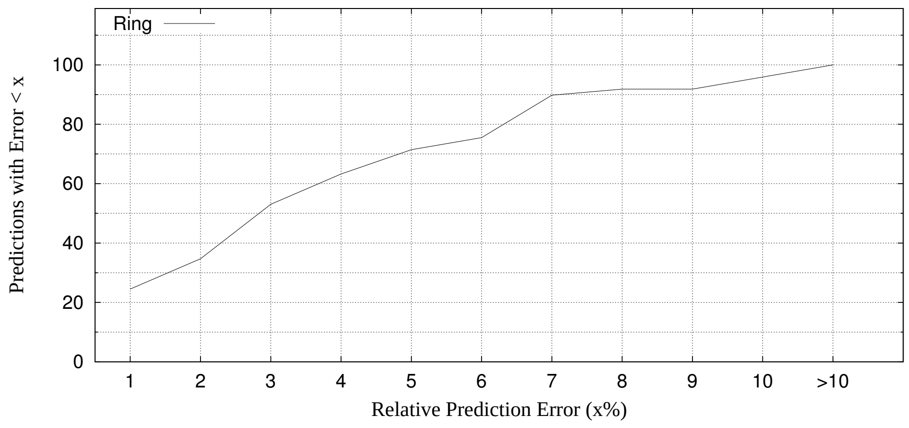
(c) Chain

Figure 6.6: Commutative predicted error of throughput of PBFT, Quorum, and Chain.

small dimension (around 400 tuples), especially, since BFT protocols always have some margin of fluctuations. This urges us to choose a prediction function with smooth curves; otherwise, having a prediction function that passes through every actual throughput value will lead to over-fitting, and thus the prediction error increases again while predicting new unknown data.



(a) Zlight



(b) Ring

Figure 6.7: Commutative predicted error of throughput of Ring and Zlight.

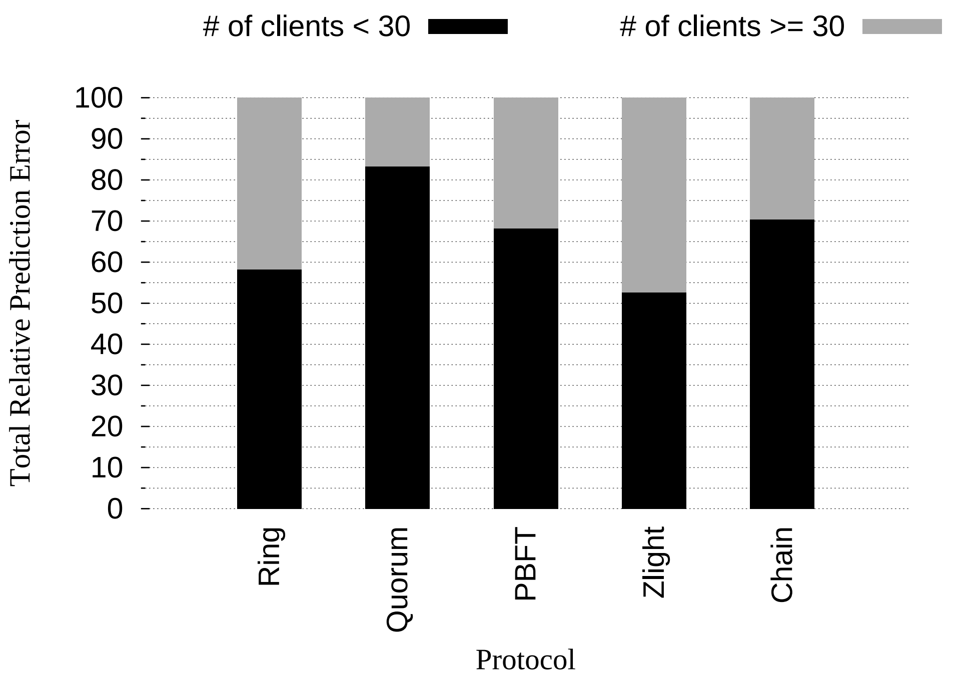
6.6.7 The Effect of Impact Factors

In this section, we discuss the effect of the impact factors on prediction. Figure 6.8 conveys the prediction error as the number of clients and the request size change.

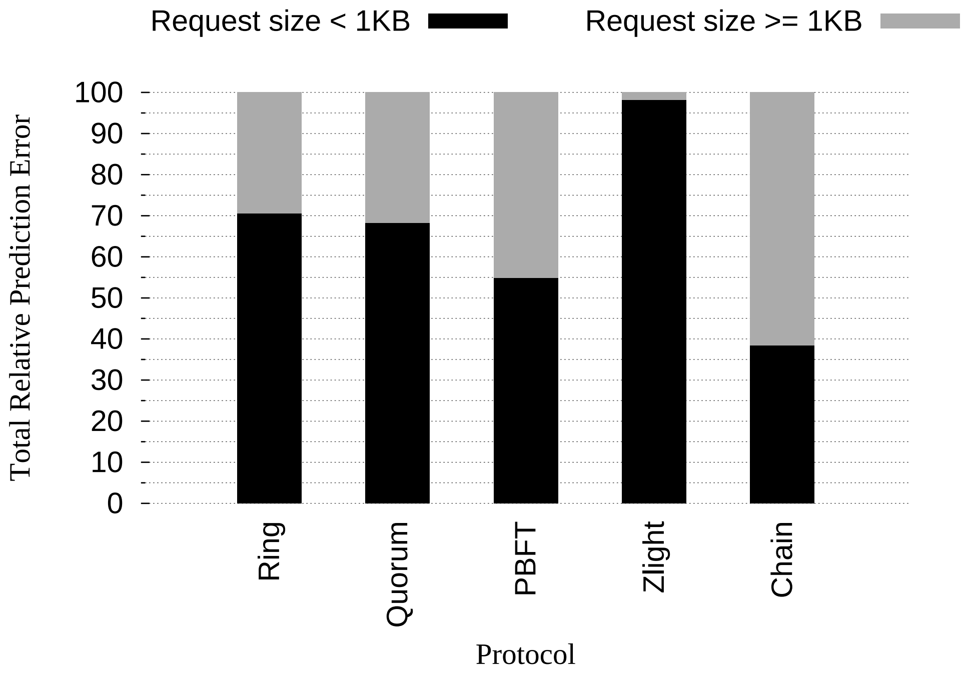
Subfigure 6.8(a) conveys the prediction where the number of clients is less or more than 30 clients. The figure depicts the fact that prediction accuracy is larger with more than 30 clients in all protocols. We expected this since the protocols become stable with more clients and hence prediction becomes easier.

Two interesting observations can be noticed in this graph. First, notice that in Quorum and Chain, prediction errors are more frequent with fewer clients (83% and 70%, respectively). This is referred to the quick change in the throughput (i.e., increasing accelartion) with few clients. With a high number of clients, however, these protocols exhibit a stable throughput, and hence prediciton errors decrease significantly. Second, the prediction errors of Zlight are almost 50% either above or below 30 clients. We refer this to the uniform change of the throughput of Zlight as more clients access the system; this yields prediction errors even with high number of clients (though we expect that this percentage will be somehow similar to other protocols incase Zlight is working with no issues).

On the other hand, Subfigure 6.8(b) conveys that the request size has a great impact



(a) Impact of number of clients



(b) Impact of request size.

Figure 6.8: Effect of impact factors on prediction accuracy.

on the accuracy of prediction. The figure shows that the prediction error when the request size is less than 1KB is almost twice that of sizes larger than 1KB. This is expected since with larger request sizes the network becomes more stable and thus throughput is less fluctuating.

The prediction result of Zlight in Subfigure 6.8(b) is surprising; predictions appear to be dramatically affected by small request sizes. In fact, we noticed that the real throughput results of Zlight are more fluctuating than the other protocols when the request size is small, this what is making the prediction error too large with small requests. Zlight becomes stable with large payloads and consequently prediction becomes more accurate. Similarly, the values of Chain are also surprising; since one would expect fewer prediction errors as messages become larger. In fact, upon observing the results of Chain, we noticed that this variation is due to the prediction points with few clients. With few clients and large request sizes, the throughput of Chain becomes quite changing. This could be noticed from Subfigure 6.8(a) where Chain has 70% of the errors with fewer than 30 clients.

6.6.8 Switching Overhead

In Adapt, we use the same switching technique in Chapter 5. In addition, the results are quite close to those in Subsection 5.3.2, in the same chapter, with a slight difference between protocols due to the difference in their message headers. Thus, not to repeat ourselves, we urge the reader to have a look at Subsection 5.3.2 for more details.

In addition, we differentiate between two cases in Adapt where selection is performed: during system normal operation, and upon failure detection. In the former case, event triggering, prediction, and evaluation are done while the system is running. Thus, they do not impose any additional delay overhead since switching has not yet occured. If Adapt took the decision to choose another protocol, then switching happens and imposes the cost that was discussed in Chapter 5. In the latter case, where some failure is detected, the replicas stop handling any request until switching happens (maybe switching to PBFT since it maintains failures). Therefore, evaluation must be done while switching. This imposes an additional overhead, i.e., the cost of selection process. We did not measure this overhead since we did not finish the Event System yet. However, our prediction scripts show that the time could not exceed 1 second, which is considered small when there are big differences among replicas as we explore in the following sections.

6.7 Worthy Switching

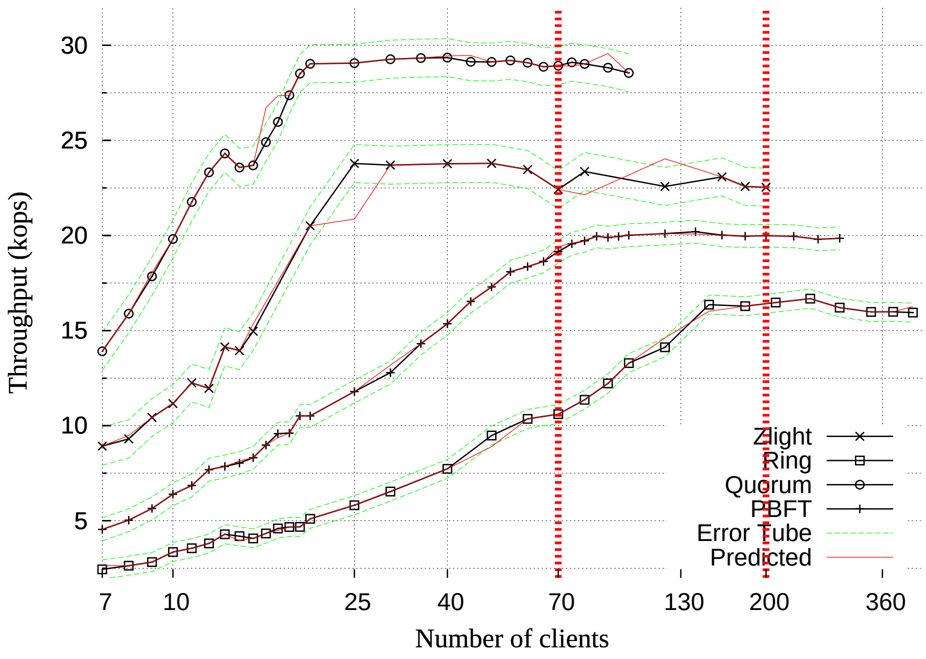
As we have seen in Chapter 4, switching is only worthy if:

$$E_{max} - E_{curr} \geq S_{thr} \tag{6.6}$$

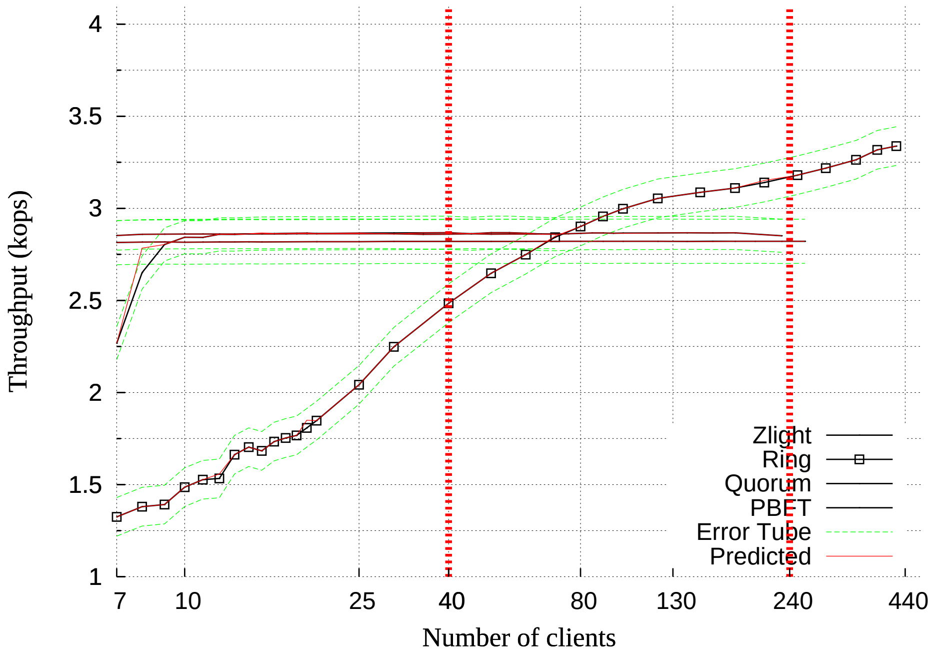
where E_{max} and E_{curr} in this equation are the evaluation scores corresponding to the preferred protocol (under the current state s_t) and the currently running one, respectively. S_{thr} represents the threshold over which switching is considered worthy. S_{thr} is a decision choice of the service administration or the user (e.g., 10%).

In the modes DYNAMIC and HEURISTIC of the selection model that was introduced in Chapter 4, the system switches to a new protocol that is evaluated as the 'preferred' protocol under the new system state. To recall the details, you may see Section 4.6 in Chapter 4. In this section, however, we focus on discussing the method experimentally.

To understand the mechanism, we provide some graphs in Figure 6.9 that can describe the results well. The graphs convey the throughput of Redis application using two different payload message sizes: 0KB and 4KB, respectively. In the experiment that corresponds to the second graph, the network had a higher contention than the former. Note that, we have noticed a lot of interesting cases to discuss; however, we have chosen these two since the difference among protocols is clear, and the switching cases are quite reasonable.



(a) 0KB request size



(b) 4KB request size

Figure 6.9: Real and predicted throughput with the error tubes of BFT protocols.

6.7.1 Experimental Switching

The x-axis of the graphs represents the number of clients, whereas the y-axis represents the achieved throughput of the protocol. The graphs convey the curves of 4 protocols: PBFT, Quorum, Zlight, and Ring. Their corresponding graphs are colored in black, and are assigned different linepoints to identify them clearly. The corresponding predicted curves are plotted in red on each curve. They are almost confounded, most of the time, with the real curve, however, they skew sometimes when the prediction error of some point is high (e.g., the curve of Zlight in Subfigure 6.9(a) for 25 and 120 clients). This can be resolved with further training. The corresponding green curves around each real (black) curve resemble the width of the error tubes. The error tubes are simply the original (black) graph with adding/subtracting the average prediction error. Thus, if the average prediction error of a protocol is ϵ, then the width of the error tube is 2ϵ. In reality, an error tube has a variable width versus the number of clients, however, we used the average error in these graphs for clarity. Of course, using a variable error tube is much accurate, thus it makes our justification stronger.

First, lets analyze Subfigure 6.9(a) first. In this graph, we notice that the throughput discrepancy among protocols is significant. For instance, look at the difference between Ring and Quorum. The latter is sometimes 5 times better than Ring. The result is similar in the cases of Zlight and PBFT, but with smaller variation. In fact, this is referred to the short messaging pattern of Quorum and the minimal authentications needed [6].

Moreover, the width of the error tubes can be clearly observed. The graph shows that this width is not equal for all protocols (this was explained in previous sections). Note that, the width of the error tubes ranges between 1Kops and 2.5Kops in our case. Thus, for any difference in throughput between two protocols, half of the two error tube widths should be subtracted. For instance, consider PBFT and Quorum with error tube widths 1.5Kops and 2Kops, respectively; the difference between the throughputs of these protocols should be: $\delta = (1.5Kops + 2Kops)/2 = 1.75Kops$. A closer look at the corresponding graphs conveys that the difference in throughout among the two graphs is large, and ranges from 10Kops to 15Kops. Subtract the error tubes, we get the difference ranges from $8.25Kops$ (i.e., $10Kops - 1.75Kops$) to $13.25Kops$ (i.e., $15Kops - 1.25Kops$). Assume the client have chosen a threshold of 10%, thus, it is: $10 \times 30Kops/100 = 3Kops$ (considering the peak throughput of Quorum). Since the threshold $3Kops$ is quite less than $8.25Kops$ and $13.25Kops$, i.e., the difference in their throughput, then it is strictly worhty to switch from PBFT to Quorum, of course, if the other conditions are also satisfied according to the selection model. The same logic can be applied to the other protocols too.

An interesting case in Subfigure 6.9(a) is the difference between PBFT and Zlight between 70 clients and 130 clients (notice the vertical dotted red lines). In this region, the width of the error tubes is 1.5Kops and 2.5Kops for PBFT and Zlight, respectively. The difference δ becomes: $\delta = (1.5Kops + 2.5Kops)/2 = 2Kops$. Since in this region the corresponding throughputs of the protocols are almost around 20Kops and 23Kops, respectively; then using the same threshold as above, 10% (i.e., $23Kops \times 10/100 = 2.3Kops$), we get the total difference: $23Kops - 20Kops - 2Kops = 1Kops$. Now, since this difference is smaller than the threshold, i.e. 2.3Kops, then we say that in this region

[6] To focus on the goal of this section we do not explain why this difference is achieved. On the contrary, this was thoroughly discussed in Chapter 3.

switching is not worthy and, thus, even if the evaluation is done, the running protocol is kept running (unless other factors forced switching according to the model).

Here, we would like to explore two limitations of Adapt. In the above example, notice that in reality we remove the error tubes, then the above difference becomes: $23Kops - 20Kops - 0Kops = 3Kops$ which is greater than the threshold 2.3Kops; thus switching should occur. However, we have seen that switching is considered not worthy in such case in Adapt (due to the prediction error). Another limitation is the points that skew significantly from the curve due to non accurate predictions. This would give a wrong decision about whether switching is worthy or not as compared to reality if the throughout of the protocols is reasonably close.

The same logic can be applied to Subfigure 6.9(b). The graph depicts the throughput of the protocols where 4KB payloads are used. The throughput of the protocols reach the peak throughput very soon (with 10 clients) due to the large payloads (i.e., reaching the network bottleneck). Consequently, the curves take the shape of a plateau. Ring is an exception since it allows any replica to receive requests and, hence, it circumvents to exceed the network limitation. Therefore, as depicted in the graph, the throughputs of PBFT, Zlight, and Quorum become equivalent with more than 10 clients. The throughput of Ring is much lower than the other protocols; however from 80 clients and more, Ring becomes dominant.

In this case, switching is only worthy between Ring and the other protocols since Zlight, Quorum, and PBFT are almost equivalent in throughput. For a 10% switching threshold, i.e., 0.34 Kops (considering the peak throughput of Ring), we notice that switching between Ring and the other protocols is not worthy when the number of concurrent clients is between 40 and 240 clients (see the red dotted vertical lines). Of course, we always consider the distance between the error tubes and not the actual curves. The logic is the same as the previous graph, thus we keep further obeservations to the reader.

6.8 Adapt Dynamic Mode

In this section, we return back to discuss the behavior of Adapt in the Dynamic mode that was presented in Chapter 4. We compare it with the Static mode and we show how it is more efficient. But here, we focus on the application of the selection model, experimentally. Since we used the prediction method for the protocols: PBFT, Zyzzyva (i.e., Zlight), Quorum, Ring, and Chain, then we confine our examples on these protocols. By this, we guarantee to provide clear examples with fruitful results.

6.8.1 Recall Model Formulas and Examples

Let us first recall the formulas used in the selection model. The evaluation score E is as follows:

$$\begin{cases} E = D \circ C \\ where\ D = \left\lfloor \frac{1}{a}.A \ \dot{\vee} \ (e_a - U) \right\rfloor \\ and\ C = B^{\pm}.(V \circ W). \end{cases} \tag{6.7}$$

The β KPI values of matrix $B^{\pm}$ are calculated as follows:

$$\begin{cases} \beta_{ij}^{+} = 1 - \dfrac{max_i - \beta_{ij}}{max_i - min_i}; \\ \beta_{ij}^{-} = 1 - \dfrac{\beta_{ij} - min_i}{max_i - min_i}; \\ where\ i \leq b\ and\ j \leq n. \end{cases} \tag{6.8}$$

And finally, the preferred protocol to be launched in the next state is chosen as follows:

$$p_{next} = p_i,\ s.t.\ E_{i,s} = max(E) = \max_{1 \leq j \leq n} E_{j,s}. \tag{6.9}$$

Now, we concisely try to recall the example used in the discussion of the Dynamic mode in Chapter 4. After removing the non considered protocols for prediction, we get the table of analytical values of β-KPIs is:

	PBFT	Zyzzyva	Quorum	Ring	Chain
Throughput (β^{+})	**1**	2	**5**	1.25	3.33
Latency (β^{-})	4	3	**2**	**9**	5
Capacity (β^{+})	6	7	**1**	**10**	8

Table 6.2: Analytical values of β^{+} and β^{-} for selected state-of-the-art BFT protocols.

Next, we recall the matrix $B^{\pm}$ used before in the following:

$$B = \begin{pmatrix} 1 & 4 & 6 \\ 2 & 3 & 7 \\ 5 & 2 & 1 \\ 1.25 & 9 & 10 \\ 3.33 & 5 & 8 \end{pmatrix} \text{ and then: } B^{\pm} = \begin{pmatrix} 0 & 0.71 & 0.55 \\ 0.25 & 0.86 & 0.67 \\ 1 & 1 & 0 \\ 0.06 & 0 & 1 \\ 0.58 & 0.57 & 0.78 \end{pmatrix}$$

For clarity of presentation, we assume that the discrete matrix $D = e_a$, i.e., any protocol is allowed for comparison, and we focus on the continuous matrix C. To conduct a fair comparison, we use the same matrices V and W that were used in Example 4.10 in Chapter 4. Thus, we use the following matrices:

$$V = \begin{pmatrix} 3 \\ 3 \\ 4 \end{pmatrix}; and\ W = e_b = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}; \text{ and } D = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix}.$$

The evaluation score E in this case (after excluding the non predicted protocols) becomes:

$$E = \begin{pmatrix} 4.33 \\ 6.01 \\ 6 \\ 4.18 \\ 6.57 \end{pmatrix}$$

Thus, according to this example, Chain is always chosen as the ‘preferred’ protocol.

6.8.2 Applying Dynamic Mode

In this section, we try to apply the Dynamic mode of the selection model since we have now run time predictions of KPIs. First, we start by writing down the tables of β-KPIs in the dynamic mode. Since here KPI values are derived at run time through prediction, then we choose three interesting system states S_0, S_1, and S_2 not to limit the study but, as usual, choosing the most interesting cases and for clarity reasons. According to our experiments, the tables used to plot matrix B are in Multi-table 6.3:

[S_0: Predicted β KPIs with 10 clients, 128B request size.]

	PBFT	Zyzzyva	Quorum	Ring	Chain
Throughput (β^+)	6390.59	11160.5	**19814.2**	**3357.44**	7016.72
Latency (β^-)	1.25e-03	8.85e-04	**5.01e-04**	**2.86e-03**	1.31e-03
Capacity (β^+)	6	7	**1**	**10**	8

[S_1: Predicted β KPIs with 40 clients, 128B request size.]

	PBFT	Zyzzyva	Quorum	Ring	Chain
Throughput (β^+)	15381.7	**23771.8**	-	**7725.37**	21018.7
Latency (β^-)	2.26e-03	**1.66e-03**	-	**5.05e-03**	1.67e-03
Capacity (β^+)	**6**	7	-	**10**	8

[S_2: Predicted β KPIs with 400 clients, 4KB request size.]

	PBFT	Zyzzyva	Quorum	Ring	Chain
Throughput (β^+)	2851.29	2821.15	-	**3337.85**	**2715.49**
Latency (β^-)	**8.01e-02**	9.18e-02	-	1.25e-01	**1.35e-01**
Capacity (β^+)	**6**	7	-	**10**	8

Table 6.3: Predicted values of β KPIs for selected state-of-the-art BFT protocols.

The throughput in the above tables is predicted. The value of latency is the real one. The capacity is kept as before. Note that using some non predicted data do not change the results since prediction accuracy is more than 95%. In additon, we replaced Quorum values by ‘-’ when it does not work (since it does not tolerate a high number of clients). The bolded areas correspond to the max and min values.

Now, the β^+ and β^- values can be calculated according to the formulas in Equation 6.8. Then we write down the matrix $B^{\pm}$ for the three states S_0 S_1, and S_2 directly as follows:

$$B^{\pm}_{S0} = \begin{pmatrix} 0.18 & 0.68 & 0.55 \\ 0.47 & 0.84 & 0.67 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0.22 & 0.56 & 0.78 \end{pmatrix}$$

$$B^{\pm}_{S1} = \begin{pmatrix} 0.48 & 0.82 & 0.55 \\ 1 & 1 & 0.67 \\ - & - & 0 \\ 0 & 0 & 1 \\ 0.82 & 0.99 & 0.78 \end{pmatrix}$$

$$B^{\pm}_{S2} = \begin{pmatrix} 0.21 & 1 & 0.55 \\ 0.17 & 0.79 & 0.67 \\ - & - & 0 \\ 1 & 0.18 & 1 \\ 0 & 0 & 0.78 \end{pmatrix}$$

We discuss each of the three states separately in the following sections.

6.8.2.1 State S0

In state S0, the calculation of the matrix $C = B^{\pm}.(V \circ W)$ is follows:

$$\begin{aligned} C00 &= B^{\pm}_{S0}.(V \circ W) \\ &= \begin{pmatrix} 0.18 & 0.68 & 0.55 \\ 0.47 & 0.84 & 0.67 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0.22 & 0.56 & 0.78 \end{pmatrix} . \begin{pmatrix} 3 \\ 3 \\ 4 \end{pmatrix} \circ \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 4.78 \\ 6.61 \\ 6 \\ 4 \\ 5.46 \end{pmatrix} \end{aligned}$$

Thus, without considering the discrete matrix D, we get that the prferred protocol according to the user preferences is Zyzzyva. This is different to Example 4.10 in Chapter 4 where Chain was chosen. The main reason is that we calculated the B values using the anlaysis Table 6.2 which are not realistic; e.g., we have seen here that Chain does not achieve a high throughput with few clients in prediction (and real) case, however, this is not mentioned in the table.

Heuristic Mode. To achieve better results, we can use the heuristic mode. This mode is similar to Dynamic mode, however, we use the matrix W to change the user preferences according to system defined heuristics. In the above state S0, the system notices that the number of clients is 10, which is small. Then the system, in this case, uses the heuristic rule that says 'no contention ==> latency is more important than throughput and capacity'. Hence, we use the following matrix:

$$W1 = \begin{pmatrix} 3 \\ 6 \\ 1 \end{pmatrix}$$

The matrix C is then as follows:

$$\begin{aligned}
C01 &= B^{\pm}_{S0}.(V \circ W1) \\
&= \begin{pmatrix} 0.18 & 0.68 & 0.55 \\ 0.47 & 0.84 & 0.67 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0.22 & 0.56 & 0.78 \end{pmatrix} . \begin{pmatrix} 3 \\ 3 \\ 4 \end{pmatrix} \circ \begin{pmatrix} 3 \\ 6 \\ 1 \end{pmatrix} \\
&= \begin{pmatrix} 0.18 & 0.68 & 0.55 \\ 0.47 & 0.84 & 0.67 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0.22 & 0.56 & 0.78 \end{pmatrix} . \begin{pmatrix} 9 \\ 18 \\ 4 \end{pmatrix} = \begin{pmatrix} 16.06 \\ 22.03 \\ 27 \\ 4 \\ 15.18 \end{pmatrix}
\end{aligned}$$

Therefore, the preferred protocol according to the heuristic mode on state S0 is Quorum (the corresponding evaluation score is 27), which is more reasonable than before. since Quorum achieves the best latency and throughout in contention-free conditions.

6.8.2.2 State S1

Let us move now two state S1, we calculate the matrix C1:

$$\begin{aligned}
C1 &= B^{\pm}_{S1}.(V \circ W) \\
&= \begin{pmatrix} 0.48 & 0.82 & 0.55 \\ 1 & 1 & 0.67 \\ - & - & 0 \\ 0 & 0 & 1 \\ 0.82 & 0.99 & 0.78 \end{pmatrix} . \begin{pmatrix} 3 \\ 3 \\ 4 \end{pmatrix} \circ \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 6.1 \\ 8.68 \\ - \\ 4 \\ 8.55 \end{pmatrix}
\end{aligned}$$

In this state S1, however, the preferred protocol becomes Zyzzyva. This makes sense, since Zyzzyva achieves better performance than other protocols as we have seen in previous sections (e.g., see Figure 6.9). Thus switching will occur in this state to Zyzzyva. Note that, here we can use heuristics too, but we did not define a heuristic rule for this case. We recall that the definition of heuristics is a future plan.

6.8.2.3 State S2

The calculation of the matrix $C2$ in the next state S2 is done similarly:

$$\begin{aligned}
C20 &= B^{\pm}_{S2}.(V \circ W) \\
&= \begin{pmatrix} 0.21 & 1 & 0.55 \\ 0.17 & 0.79 & 0.67 \\ - & - & 0 \\ 1 & 0.18 & 1 \\ 0 & 0 & 0.78 \end{pmatrix} . \begin{pmatrix} 3 \\ 3 \\ 4 \end{pmatrix} \circ \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix} = \begin{pmatrix} 5.83 \\ 5.56 \\ - \\ 7.54 \\ 3.12 \end{pmatrix}
\end{aligned}$$

Therefore, in state S2, where the network is highly contended, we notice that Ring is chosen as the preferred protocol (that corresponds to the 4^{th} entry of value 7.54). The result is reasonable since Ring exceeds the network limitations of the other protocols. In fact, the same result can be seen clearly in Figure 6.9 too.

Heuristic Mode. In the above state S2, we can use the heuristic rule 'very high contention ==> capacity is more important than latency and throughput'. Hence, we use the following matrix:

$$W2 = \begin{pmatrix} 3 \\ 1 \\ 6 \end{pmatrix}$$

The matrix C is then as follows:

$$\begin{aligned}
C22 &= B^{\pm}_{S2}.(V \circ W2) \\
&= \begin{pmatrix} 0.21 & 1 & 0.55 \\ 0.17 & 0.79 & 0.67 \\ - & - & 0 \\ 1 & 0.18 & 1 \\ 0 & 0 & 0.78 \end{pmatrix} . \begin{pmatrix} 3 \\ 3 \\ 4 \end{pmatrix} \circ \begin{pmatrix} 3 \\ 1 \\ 6 \end{pmatrix} \\
&= \begin{pmatrix} 0.21 & 1 & 0.55 \\ 0.17 & 0.79 & 0.67 \\ - & - & 0 \\ 1 & 0.18 & 1 \\ 0 & 0 & 0.78 \end{pmatrix} . \begin{pmatrix} 9 \\ 3 \\ 24 \end{pmatrix} = \begin{pmatrix} 18.09 \\ 19.98 \\ - \\ 33.54 \\ 18.72 \end{pmatrix}
\end{aligned}$$

In this heuristic case, the result remains the same, i.e., choosing Ring as the 'preferred' protocol which is trivial. However, notice that the other scores in C22 are more reasonable than C20. For instance, notice that the evaluation score of Ring without heuristics (i.e., 7.54) is somehow close to the other values in C20. In the heuristic mode C22, however, the score of Ring is far away from all protocols which gives a strict selection. The result is true in the former case, but the latter gives more confidence. In addition, the evaluation score of Chain in C20 is very small as compared to other protocols. This does not sound logical since Chain achieves a good performance in reality, that is close to Zyzzyva under contention. Interestingly, this appears in the heuristic mode, in C22. Again, though the preferred protocol has not changed, the result sounds more realistic.

6.9 Conclusion

We introduced *Adapt*, a new BFT system designed to boost up the performance of BFT protocols, however, this time without introducing a new protocol; we rather collect existing BFT protocols in one package, we assess their run time performance under given system conditions, and we launch the protocol with the best performance under these conditions.

First, we explored the different modules of Adapt. Adapt is composed of: BFT Subsystem (BFTS), Event Subsystem (ES), and Quality Control Subsystem (QCS). We focused on the QCS. We have presented a mechanism using SVR [21] to predict the run time performance of the protocols. Then, we studied the prediction of throughput, and we showed that prediction accuracy ranges between 95% and 98%. We explained some prediction observations and conducted discussions to see the effect of the *Impact Factors* on prediction, namely: the number of clients and request size. We performed a concrete experimental study demonstrating some cases that are worthy to switch between protocols. Finally, we demonstrated how to use the Dynmic and Heuristic modes of the selection model presented in Chapter 4. The discussion yielded very reasonable results that are close to reality.

In the future, we plan to examine Adapt in the presence of heterogeneous payloads, and to study further impact factors on the performance of BFT protocols (to include them in the evaluation). In addition, we plan to present an event subsystem ES to accomplish all the modules of Adapt, and to define a set of heuristics for the Heuristic mode.

Chapter 7

Conclusions

The enormous reliance of our society on computers in personal life and business demands increasing reliability. On the other hand, the evolution of computer technologies, programming languages, and platforms raised up new failures and bugs that are often classified as Byzantine (or arbitrary) faults. Consequently, Byzantine fault tolerance is becoming crucial to resolve this sort of failures. Although notable research has been conducted to achieve BFT protocols that are convincing to practitioners, existing BFT protocols are clearly at a far distance from the one-size-fits-all protocol. Our work makes this distance closer, and helps researchers to focus on other BFT problems like scalability and practical issues.

The theoretical analyses of existing BFT protocols convey a significant discrepancy among them. However, this variety is quite useful as distinct BFT users have different demands. Thus, while a specific protocol is recommended by some user, other users may prefer another protocol that suites their demands better. Since dozens of protocols have been introduced so far, users may get bothered while making their selections. In our thesis, we provided an automated approach that can make the selection process simple and fast. Our idea is to match the user preferences with the profiles of the different BFT protocols. Then, users can choose the best match. The selection mechanism works in Static, Dynamic, and Heuristic modes. With the relaince of current applications on Web Services and cloud computing, we beleive that our work will add a value to these services by making deployed services using such techniques more reliable.

In the static mode, users are allowed to choose a single protocol only. However, the recovery phase of existing protocols is expensive when compared with the trivial phase of the protocol itself during free-failure. In services that are usually stable but encounter alternating periods of failure, the overhead of the recovery phase can be greater. In this thesis, we developed a new technique to maintain the same anticipated performance of the protocol even during recovery. The idea is to let the protocol recover to itself again, this time, using another set of backup replicas. When the time to repair a faulty replica is big, the profit induced by our approach becomes significant. In general, our mechanism improves the performance of existing protocols during failures whether the selection model is used or not.

The above approach is useful with static selection since the user is convinced with the chosen protocol and he/she has a steady-state system. However, when the system state is too fluctuating, the situation becomes more complex. For instance, consider a service that endure different conditions during the same day or week. No single protocol will be convincing to the user anymore. A dynamic solution becomes crucial in such situation.

We introduced Adapt as a solution. Adapt can adapt to the changes of the underlying system at run time. Adapt (1) predicts the performance of the protocols while running, (2) decides which is the preferred protocol under the new conditions, (3) and tells whether switching to the selected protocol is worthy or not. Moreover, we used some heuristics in Adapt to optimize the performance of the system further.

The adaptability of BFT protocols to underlying conditions at run-time is novel. Though BFT requires further solutions on the scalability and practical sides, we believe that our work ends the competition of BFT protocols performance-wise, and allows research community to focus on solving the former issues. At the same time, we believe that further research should be conducted on adaptable BFT, especially, for the dynamic and heuristic modes.

7.1 Summary

This section summarizes the major points of the thesis.

7.1.1 Theoretical and Experimental Analysis

In this thesis, we conducted three analyses to understand the behavior and properties of BFT protocols. The first addressed the different characteristics of BFT protocols. The second (a traditional one) discussed the impact of message exchange patterns and message authentications on performance. The third is an experimental analysis that showed the discrepancy in performance among BFT protocols. The latter analysis was accompanied with a justification using Queuing theory. The three analyses together demonstrated how one-size-fits-all BFT protocols are hard to achieve.

7.1.2 Selection Mechanism

Then, we introduced a BFT selection model and selection algorithm to choose the 'preferred' protocol out of a collection of candidate protocols. The 'preferred' protocol for a specific user is the closer to his demandes. Each protocol is associated with a profile. The profile comprises (1) the properties that differentiate this protocol, (2) its required assumptions, and (3) its performance metrics. A user provides his preferences that are matched against the profiles of candidate BFT protocols through an evaluation process. The protocol that corresponds to the highest evaluation score is considered the 'preferred' protocol for this specific user. Selection is done through establishing a set of mathematical formulas to automate the evaluation process.

The model works in three modes: Static, Dynamic, and Heuristic.

7.1.2.1 Static Mode

The static mode is designed for systems that often have a steady behavior, i.e., they do not encounter a lot of ups and downs in their state. Thus, a single chosen protocol can always be satisfactory for such systems. The situation is very useful in Web Services and Clouds where cloud vendors can sell BFT as a service (signed in the SLA contract) to provide services with Byzantine resiliency. Our approach provides a way for users to choose the preferred protocol for their service. Users have to provide their preferences, and the service provider runs the algorithm to match user preferences against the profiles of the candidate protocols it supports. We have driven a detailed discussion and examples on the matching process to show its automated nature and simplicity. In addition, we demonstrated how this approach can yield decision choices that match reality.

7.1.2.2 Re-Abstract Family

In addition, we introduced a BFT Re-Abstract family of protocols based on abortable BFT to improve the performance of existing protocols during recovery. Re-Abstract family members are useful in the Static mode. The fact is that most protocols suffer from degraded performance upon failure, i.e., during the recovery phase. Our approach solved this problem by having two sets of replicas: $3f+1$ Active replicas and f additional Passive replicas. The protocol, i.e., an Abstract, operates as usual on the Active replicas, while the f Passive replicas are idle. Upon failure detection, the system aborts and switches back to itself after it rules out the *suspicious* replicas. Suspicious replicas are either Byzantine or slow. We presented a mechanism to identify the suspicious replicas and then replace them with *correct* Passive ones. We have shown that our approach conserves the actual performance of the original Abstract during recovery. In fact, the throughput can be improved up to two folds in some cases. Furthermore, the evaluation conveyed that Re-Abstract family members are efficient when the repair time of failed replicas is not short. On the contrary, Re-Abstract is not quite useful if the failure is transient (e.g., for less than one second).

7.1.2.3 Dynamic and Heuristic Modes

The Dynamic mode addresses systems that are quite fluctuating, for instance: variable contention, message payloads, malicious clients, etc. In this situation, it is necessary to move to another protocol that better fits the new system conditions. Adapt, a novel BFT system, operates in the Dynamic mode to switch between protocols at run time according to the changes in the underlying system state. Adapt collects existing protocols and libraries in a pool, called the BFT system (BFTS). An Event Subsystem (ES) monitors the communication, and can send an event to the Quality Control Subsystem (QCS) triggering an evaluation process for the protocols. QCS runs a prediction phase to assess the performance of all protocols (e.g., throughput and latency) on the fly, and then runs the selection algorithm to cull the most suitable protocol for the next phase. We used a well-known prediction mechanism in Machine Learning called Support Vector Machines for Regression (SVR). In SVR, prediction is based on training a prediction function on an experimental knowledge base. This is important in complex communication systems since it is hard to divine the performance of different protocols via theoretical or analytic forecast.

The Heuristic mode is sometimes needed to adjust the preferences of the user. For instance, choosing a protocol that tolerates a high number of clients is not convenient if the system is not contended. Another example is when a user seeks low latency and high throughput but the system is flooded with requests from numerous clients. In this case, tolerating a higher number of clients becomes prior to latency and throughput. The heuristic mode uses some predefined rules that reacts with such situations and adjust the user preferences to choose the 'preferred' protocol. Our evaluation showed that this approach can sometimes be very useful.

We evaluated Adapt in different ways. First, we conveyed that the prediction accuracy of throughput ranges between 95% and 98% (considering two impact factors: number of clients and request size). Then we demonstrated how the impact factors are affecting prediction accuracy. Then we explored how using the dynamic mode yields intersting realsitic choices. In addition, we conveyed some interesting cases where switching can be very benificial. In addition, we have shown how to use the heursitic mode in Adapt, and that this is useful in many cases, especially when a BFT user has no big experience.

Though we have shown that switching in Abortable BFT is not expensive (in Chapter 5), we believe that selection process and event triggering impose some overhead in some cases. We plan to measure this in the future as soon as we finish our research on the Event Subsystem.

7.1.3 Miscellaneous

Our experiments were conducted on Emulab using two applications Redis and OpenLDAP. Also, we used LIBSVM library for prediction. On the other hand, most of our implementations were coded in C/C++ and Bash scripts.

7.2 Future Work

Our schedule comprises three major subjects and some minor ones.

First, we think that the idea of Adapt is promising and practical and hence deserves further research. For instance, the Event Subsystem (ES) is mandatory for Adapt in order to take good decisions at the right time. ES should send events about any potential changes in the underlying system. This triggers the selection and evaluation process. ES should take into consideration network factors, payload factors, contention factors, application specific factors, etc. In BFT, we think that this is not an easy task since the event should be shared among replicas, thus, avoiding Byzantine behaviors in centralized solutions.

In addition, the conveyed prediction accuracy of Adapt is high. Due to our close deadline, we were unable to consider further impact factors and KPIs in the evaluation process. In the future, we plan to include these issues in order to discover the efficiency of Adapt under more complicated situations with multiple factors.

We have seen that Adapt can use some heuristic to adjust the selection process. In this thesis, we used an example of two heuristics. We believe that defining a heuristic set of rules will be an interesting future work.

Regarding the minor subjects, we plan to conduct some experiments to study new classes of protocols, e.g., separating agreement form execution [56]. In addition, we need to find a method that can better evaluate selection in the Static mode. Currently, we evaluated the approach using human reasoning which is not constant among humans.

In parallel to this plan, we are working on other fault tolerance issues that, currently, have a growing importance. For instance, ensuring anonymous communication in the BAR model (i.e., Byzantine, Altruistic, Rational) with practical performance is still an open problem, and thus requires intensive research. We try to contribute in finding some solutions.

List of Figures

List of Tables

Bibliography

[1] Rachid Guerraoui, Nikola Knezevic, Vivien Quema, Marco Vukolic. Stretching bft. Technical Report EPFL-REPORT-149105, EPFL, 2011.

[2] Michael Abd-El-Malek, Gregory R. Ganger, Garth R. Goodson, Michael K. Reiter, and Jay J. Wylie. Fault-scalable byzantine fault-tolerant services. *SIGOPS Oper. Syst. Rev.*, 39(5):59–74, 2005.

[3] Ivo Adan and Jacques Resing. *Queueing Theory.* Eindhoven University of Technology, Eindhoven, The Netherlands, February 2002.

[4] Ali Shoker and Jean-Paul Bahsoun. Recover to Self: Re-Abstract Family (regular paper). In *The International Conference on Computer and Management*, CAMAN. IEEE, March 2012.

[5] Ali Shoker and Jean-Paul Bahsoun. Towards Byzantine Resilient Directories (to appear). In *The 11th IEEE International Symposium on Network Computing and Applications*, NCA'12. IEEE Computer Society, August 2012.

[6] Peter A. Alsberg and John D. Day. A principle for resilient sharing of distributed resources. In *Proceedings of the 2nd international conference on Software engineering*, ICSE '76, pages 562–570, Los Alamitos, CA, USA, 1976. IEEE Computer Society Press.

[7] Amazon.com. Amazon ec2, 2012.

[8] Amazon.com. Amazon web services, 2012.

[9] Yair Amir, Brian Coan, Jonathan Kirsch, and John Lane. Prime: Byzantine replication under attack. *IEEE Trans. Dependable Secur. Comput.*, 8(4):564–577, July 2011.

[10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy Katz, Andy Konwinski, Gunho Lee, David Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. A view of cloud computing. *Commun. ACM*, 53(4):50–58, April 2010.

[11] Bernhard Boser, Isabelle Guyon, and Vladimir Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the fifth annual workshop on Computational learning theory*, COLT, pages 144–152. ACM, 1992.

[12] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *Journal of the ACM (JACM)*, 32(4):824–840, 1985.

[13] Prabir Burman. A comparative study of ordinary cross-validation, v-fold cross-validation and the repeated learning-testing methods. *Biometrika*, 76(3):503–514, 1989.

[14] Miguel Castro. Practical byzantine fault tolerance. In *Ph.D Thesis*, pages 173–186, 2001.

[15] Miguel Castro and Barbara Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Trans. Comput. Syst.*, 20(4):398–461, 2002.

[16] Chih-Chung Chang and Chih-Jen Lin. LIBSVM: A library for support vector machines. *ACM Transactions on Intelligent Systems and Technology*, 2:27:1–27:27, 2011. `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[17] Byung-Gon Chun, Petros Maniatis, Scott Shenker, and John Kubiatowicz. Attested append-only memory: making adversaries stick to their word. In *Proceedings of twenty-first ACM SIGOPS symposium on Operating systems principles*, SOSP '07, pages 189–204, New York, NY, USA, 2007. ACM.

[18] Allen Clement, Manos Kapritsos, Sangmin Lee, Yang Wang, Lorenzo Alvisi, Mike Dahlin, and Taylor Riche. Upright cluster services. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, SOSP '09, pages 277–290, New York, NY, USA, 2009. ACM.

[19] Allen Clement, Edmund Wong, Lorenzo Alvisi, Mike Dahlin, and Mirco Marchetti. Making byzantine fault tolerant systems tolerate byzantine faults. In *NSDI'09: Proceedings of the 6th USENIX symposium on Networked systems design and implementation*, pages 153–168, Berkeley, CA, USA, 2009. USENIX Association.

[20] Miguel Correia, Nuno Ferreira Neves, and Neves Paulo Veríssimo. How to tolerate half less one byzantine nodes in practical distributed systems. In *IN PROCEEDINGS OF THE 23RD IEEE SYMPOSIUM ON RELIABLE DISTRIBUTED SYSTEMS*, pages 174–183, 2004.

[21] Corinna Cortes and Vladimir Vapnik. Support-vector networks. In *Machine Learning*, pages 273–297, 1995.

[22] James Cowling, Daniel Myers, Barbara Liskov, Rodrigo Rodrigues, and Liuba Shrira. Hq replication: a hybrid quorum protocol for byzantine fault tolerance. In *OSDI '06: Proceedings of the 7th symposium on Operating systems design and implementation*, pages 177–190, Berkeley, CA, USA, 2006. USENIX Association.

[23] Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Consensus in the presence of partial synchrony. *J. ACM*, 35(2):288–323, April 1988.

[24] David K. Gifford. Weighted voting for replicated data. In *Proceedings of the seventh ACM symposium on Operating systems principles*, SOSP '79, pages 150–162, New York, NY, USA, 1979. ACM.

[25] Donald Gross, John F. Shortle, James M. Thompson, and Carl M. Harris. *Fundamentals of Queueing Theory.* Wiley-Interscience, New York, NY, USA, 4th edition, 2008.

[26] Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marko Vukolić. The next 700 bft protocols. In *EuroSys '10: Proceedings of the 5th European conference on Computer systems*, pages 363–376, New York, NY, USA, 2010. ACM.

[27] Rachid Guerraoui, Nikola Knežević, Vivien Quéma, and Marco Vukolić. The next 700 bft protocol. Technical Report LPD-REPORT-2008-008, EPFL, 2008.

[28] Rachid Guerraoui, Maysam Yabandeh, Ali Shoker, and Jena-Paul Bahsoun. Obfuscating bft. Technical Report LPD-REPORT-2011-5, EPFL, 2011.

[29] Qi He, Constantine Dovrolis, and Mostafa Ammar. On the predictability of large transfer tcp throughput. *SIGCOMM Comput. Commun. Rev.*, 35(4):145–156, 2005.

[30] Chih-Wei Hsu, Chih-Chung Chang, and Chih-Jen Lin. A Practical Guide to Support Vector Classification, 2000.

[31] IBM Corporation. Understanding ldap - design and implementation, 2004.

[32] Ramakrishna Kotla, Lorenzo Alvisi, Mike Dahlin, Allen Clement, and Edmund Wong. Zyzzyva: speculative byzantine fault tolerance. *SIGOPS Oper. Syst. Rev.*, 41(6):45–58, 2007.

[33] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978.

[34] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

[35] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems*, 4:382–401, 1982.

[36] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16:133–169, May 1998.

[37] Barbara Liskov, Sanjay Ghemawat, Robert Gruber, Paul Johnson, and Liuba Shrira. Replication in the harp file system. In *Proceedings of the thirteenth ACM symposium on Operating systems principles*, SOSP '91, pages 226–238, New York, NY, USA, 1991. ACM.

[38] N.A. Lynch. *Distributed algorithms.* Morgan Kaufmann, 1996.

[39] Sheila A. McIlraith, Tran Cao Son, and Honglei Zeng. Semantic web services. *IEEE Intelligent Systems*, 16:46–53, 2001.

[40] Brahim Medjahed, Athman Bouguettaya, and Ahmed K. Elmagarmid. Composing web services on the semantic web. *The VLDB Journal*, 12(4):333–351, November 2003.

[41] Massachusetts Institute Of, Cynthia Dwork, Nancy Lynch, and Larry Stockmeyer. Cynthia dwork and nancy lynch. *Journal of the ACM*, 35:288–323, 1988.

[42] Brian M. Oki and Barbara H. Liskov. Viewstamped replication: A new primary copy method to support highly-available distributed systems. In *Proceedings of the seventh annual ACM Symposium on Principles of distributed computing*, PODC '88, pages 8–17, New York, NY, USA, 1988. ACM.

[43] OpenLDAP Foundation. OpenLDAP Project, 2010.

[44] OpenLDAP Foundation. OpenLDAP Software 2.4 Administrator's Guide, 2010.

[45] Raluca Halalai, Thomas Henzinger, and Vasu Singh. Quantitative evaluation of bft protocols. *Quantitative Evaluation of Systems, International Conference on*, 0:255–264, 2011.

[46] Redis. Redis project, 2010.

[47] Fred B. Schneider. Implementing fault-tolerant services using the state machine approach: a tutorial. *ACM Comput. Surv.*, 22(4):299–319, 1990.

[48] Atul Singh, Tathagata Das, Petros Maniatis, Peter Druschel, and Timothy Roscoe. Bft protocols under fire. In *NSDI'08: Proceedings of the 5th USENIX Symposium on Networked Systems Design and Implementation*, pages 189–204, Berkeley, CA, USA, 2008. USENIX Association.

[49] Smola, Alex J. and Schölkopf, Bernhard. A tutorial on support vector regression. *Statistics and Computing*, 14:199–222, 2004.

[50] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, and Lau Cheuk Lung. Spin one's wheels? byzantine fault tolerance with a spinning primary. In *Proceedings of the 2009 28th IEEE International Symposium on Reliable Distributed Systems*, SRDS '09. IEEE Computer Society, 2009.

[51] Giuliana Santos Veronese, Miguel Correia, Alysson Neves Bessani, and Lau Cheuk Lung. Ebawa: Efficient byzantine agreement for wide-area networks, 2010.

[52] M. Wahl, T. Howes, and S. Kille. Lightweight directory access protocol (v3). RFC Editor, 1997.

[53] Xia Wang, Tomas Vitvar, Mick Kerrigan, and Ioan Toma. A qos-aware selection model for semantic web services. In *Proceedings of the 4th international conference on Service-Oriented Computing.* Springer-Verlag, 2006.

[54] Aaron Weiss. Computing in the clouds. *netWorker*, 11(4):16–25, December 2007.

[55] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):255–270, 2002.

[56] Jian Yin, Jean-Philippe Martin, Arun Venkataramani, Lorenzo Alvisi, and Mike Dahlin. Separating agreement from execution for byzantine fault tolerant services. *SIGOPS Oper. Syst. Rev.*, 37(5):253–267, October 2003.

[57] Hong Qing Yu and Stephan Reiff-Marganiec. A method for automated web service selection. In *Proceedings of the 2008 IEEE Congress on Services - Part I*, SERVICES '08. IEEE Computer Society, 2008.

[58] Kurt Zeilenga, Howard Chu, Pierangelo Masarati, and Others. OpenLDAP. Computer Software, 1998.

[59] Kurt D. Zeilenga. Lightweight directory access protocol (ldap) transactions. RFC Editor, March 2010.