Improving the Efficiency and Reliability of Text Messaging Gateways

Daniel Brahneborg

2022-11-22

It doesn’t matter who you are
It’s what you do that takes you far
And if at first you don’t succeed
Here’s some advice that you should heed

You get up again, over and over
You get up again, over and over
You get up again, over and over
You get up again, over and over

– Madonna

Abstract

When software communicates, it typically uses some middleware. In this thesis we focus on the group of middleware called “messaging gateways”, which normally uses the store-and-forward architecture. Our overall goal is to find ways to improve the throughput and reliability of such gateways in general, in particular the ones used for forwarding mobile text messages. In short, our work resulted in a new anomaly detection algorithm, a reduction of the number of database operations in a commercial messaging gateway, and two new highly performant data replication protocols.

In addressing the throughput, we first wanted a better understanding of how the round-trip times for outgoing requests varied, in order to correctly detect abnormal delays. This resulted in a generalized extension of exponential smoothing, which we used in a novel algorithm to detect anomalies.

We then performed an architecture analysis of an industrial messaging gateway based on its quality requirements. From this exploratory case study, we deduced a somewhat unexpected plan to migrate the balance management module to a set of microservices, and identified situations where database operations could be batched in order to also provide higher throughput.

A common and easy way to improve the reliability of a system is to replicate the stored data to one or more additional servers. However, we found that this approach often leads to lower throughput due to extra network traffic required. We addressed this by first writing a problem formulation on how the quality attributes of a messaging gateway would be affected by a multi-node configuration, resulting in a review of state of the art and state of practice for multi-node systems.

Building on this review, we developed a new data replication algorithm, which we validated in a controlled experiment. Its proof-of-concept implementation showed that even in a geo-distributed configuration, replication throughput can scale with the number of nodes.

Text messaging gateways often have a module for credit management, used when billing the senders for their traffic. In a multi-node environment the

maintenance of the corresponding credit balances requires synchronization between the nodes, for which we designed and implemented an algorithm which uses less network traffic than existing solutions in many real-world situations.

Sammanfattning

All programvara kommunicerar på ett eller annat sätt, antingen med datorns operativsystem, med annan programvara som körs på samma dator, eller över ett nätverk. Vanligtvis används någon form av mellanmjukvara för att underlätta kommunikationen, vid behov tillhandahålla protokollkonvertering och med hjälp av buffring effektivisera bandbreddsanvändningen. Modellen med avsändare → mellanmjukvara → mottagare är användbar på alla nivåer, från det fysiska länklagret till applikationslagret, där skillnaderna huvudsakligen avser detaljer.

Den här avhandlingen har fokus på applikationslagret, framför allt på gruppen mellanmjukvara kallad "messaging gateways". För att validera våra resultat använde vi en befintlig programvara skapad specifikt för att vidarebefordra SMS över hela världen. Eftersom SMS debiteras per meddelande har denna programvara också en modul för kredithantering, som ger underlag för fakturering av trafiken. Vårt övergripande mål är att identifiera olika sätt att förbättra kvalitetsattributen för sådan programvara, i synnerhet avseende prestanda och tillförlitlighet.

Till att börja med ville vi skaffa oss en bättre förståelse för variationen i svarstiderna från mobiloperatörerna, i syfte att kunna identifiera onormala beteenden. Arbetet med att nå denna förståelse resulterade i en ny variant av exponentiell utjämning och en algoritm för avvikelsedetektering. Algoritmen validerades därefter i en fallstudie.

Vidare står kraven på hög prestanda och tillförlitlighet i konflikt med varandra, eftersom hög tillförlitlighet kräver att meddelanden replikeras till en eller flera andra noder, vilket resulterar i ökad bearbetning och nätverkstrafik, och därmed påverkar prestandan negativt. Vi adresserade detta problem genom att först skriva en problemformulering för hur kvalitetsattributen skulle påverkas i en konfiguration med flera noder, vilket resulterade i en översyn av modern forskning och praxis. Därefter utvecklade vi en ny datareplikeringsalgoritm, som validerades i ett kontrollerat experiment. Resultaten från experimentet visade att även i en geografiskt utspridd konfiguration, kan prestandan öka i

takt med antalet noder.

För att slutligen säkerställa att vi framöver skapar lösningar som ger signifikanta förbättringar, utförde vi också en arkitekturanalys av SMS-programvaran. Denna fallstudie utmynnade i en något oväntad insikt om att en migrering av kredithanteringen till en uppsättning mikrotjänster skulle resultera i förbättrad prestanda för de flesta av systemets olika användarkategorier.

Popular summary

As individuals, we can choose between a plethora of systems for sending short messages. For messages sent from companies to their customers, such as meeting reminders, tickets, and authentication codes, traditional text messages are still commonly used, as this is a proven technology which works on all mobile phones. The companies usually send these messages via SMS brokers, who in turn forward them to each recipient's mobile operator. Because brokers charge the senders per message, they want to be able to handle a large number of messages for this traffic to be profitable. They also want to be sure the senders are charged the correct amount. Senders, on their part, want to be able to trust that their messages will reach the customers.

One of the software products that handle this kind of data traffic, which has some unusual features and quality requirements, is the Enterprise Messaging Gateway (EMG) from Braxo AB. Daniel Brahneborg, in a collaboration between Mälardalen University and Braxo, has built on current research to find better ways to meet the sometimes conflicting requirements of both good performance and high reliability. This has resulted in a new algorithm for finding deviations in response times, which can vary from a few milliseconds to several seconds and still be considered normal. It has also provided a more efficient technique to keep data safe when using geographically dispersed computers. A thorough analysis of EMG's architecture finally showed how its balance management could be changed to handle the steadily increasing traffic volumes of both larger and smaller SMS brokers.

Populärvetenskaplig sammanfattning

Privatpersoner kan idag välja bland många olika system för att skicka korta meddelanden mellan varandra. För meddelanden som skickas från företag till deras kunder, exempelvis mötespåminnelser, biljetter, och inloggningskoder, är dock traditionella SMS fortfarande väldigt vanliga eftersom tekniken är välbeprövad och fungerar på alla mobiltelefoner. Företagen skickar oftast dessa via SMS-mäklare, som i sin tur skickar dem vidare till mottagarnas respektive mobiloperatörer. Eftersom mäklarna tar ut en avgift per meddelande av avsändarna, vill de kunna hantera stora trafikmängder för att det ska vara lönsamt. De vill också vara säkra på att avsändarna debiteras rätt belopp. Avsändarna å sin sida vill kunna lita på att deras meddelanden kommer fram till sina kunder.

En av de mjukvaruprodukter som finns för att hantera den här sortens datatrafik, som har både speciella egenskaper och kvalitetskrav, är Enterprise Messaging Gateway (EMG) från Braxo AB. I ett samarbete mellan Mälardalens Universitet och Braxo har Daniel Brahneborg byggt vidare på aktuell forskning för att hitta bättre sätt att uppfylla de ibland motstridiga kraven av både god prestanda och hög tillförlitlighet. Arbetet har resulterat i en ny algoritm för att identifiera avvikelser i svarstider, trots att dessa kan variera från enstaka millisekunder till flera sekunder och ändå anses normala. Det har också gett en effektivare metod för att säkerhetskopiera data mellan geografiskt åtskilda datorer. En djupgående analys av EMGs arkitektur visade slutligen hur dess saldohantering skulle kunna ändras för att hantera de stadigt ökande trafikvolymerna hos både större och mindre SMS-mäklare.

Acknowledgments

Looking back on the years that have passed since I started this work, I don't think I have ever felt so lonely so often. There is "a lot to read", as Mats said with a slight grin in one of our interviews before I started, and naturally you have to do this all by yourself. You then use the new knowledge you get from all this reading, to gradually dig yourself a deeper hole, at the same time quickly reducing the number of people who understand and can relate to what you are doing until there is almost nobody left. It is lunacy. Sheer lunacy.

Therefore, first and foremost: thank you Mia, for helping me not get completely lost.

Each one of my advisors, Daniel, Wasif, Adnan, Mats, and Saad, then nudged and pushed me in various directions, gradually helping me understand how to do proper research. Thank you, all, for not giving up on me. And, perhaps even more, for not letting me quit when things got really frustrating with reviewers that seemed to have a competition in who could write the harshest rejection.

The feeling of loneliness was partly mitigated by the research school ITS ESS-H, which also helped fund this work. We actually did not meet that often, but it was a nice little community where we could see each other improve in various ways. I am glad to have met you all, even though Per and Mahshid have extra special places in my heart.

I am also thankful to Seb, Åsa, Hannes, and Lotta, who showed me that getting a PhD is possible even for mere mortals. Do not get me wrong, you are all better than most. However, as far as I know, none of you can neither fly nor walk through walls, things that seemed to be on the same difficulty level as getting a PhD.

In a recent interview, Hans-Olov Adami at Karolinska Institutet said that you should only become a researcher if there are no other alternatives. He also said that you should only do research that would never get done if you did not do it yourself. Again, looking back, that was indeed my situation. I still think that doing what is needed to get a PhD is lunacy, but I am nothing but grateful for getting the opportunity to actually try.

Gradually another feeling of community has started to become clear to me, both from everybody I met at MDU, but also from other researchers at the various conferences I attended. To be honest, I do not really understand what any one of you actually do. That is probably normal. Still, at this point I am starting to understand *how* you do it. It is like being a member of a magicians' guild, and it is quite amazing.

Daniel Brahneborg
Stockholm, 2022

Abbreviations

The abbreviations used in this thesis are provided here for convenient reference.

ATAM
Architectural Trade-off Analysis Method, a way to analyze a software architecture, with a focus on quality attributes.

CRDT
Conflict-Free, Commutative and Convergent Replicated Data Types. These are data types where the order of the applied operations has no effect on the end result. Some of these data types even allow a subset of the operations to be ignored. For example, the natural numbers $\mathbb{N}$ under the `max()` operation support both these conditions.

EMG
Enterprise Messaging Gateway, our demonstration system.

GSM
Global System for Mobile communications, a digital system for mobile telephony.

HTTP
HyperText Transfer Protocol, the primary communication protocol used for web traffic.

IA5 International Reference Alphabet number 5, a 7 bit character encoding scheme often used for mobile text messages. In the SMS domain, the character set referred to as "IA5" is however more correctly named "GSM-7".

ICAB
Infoflex Connect AB, the company that developed and maintains EMG. In 2021 this company changed its name to Braxo AB.

MPS
Messages Per Second, the unit we use for measuring throughput of a messaging system. When the incoming and outgoing throughput differ, we use the smallest of these values as the throughput of the system.

NoSQL
A common name for databases which do not use the relational paradigm of SQL databases.

PDU
Protocol Data Unit, a single data packet used for SMS traffic. Each PDU contains a login request with a username and password, a single SMS of up to 160 characters, or the acknowledgement of a received message.

QR Quality Requirements, how well a system behaves. ISO/IEC 25010 [45] expresses this in terms of performance efficiency, compatibility, reliability, etc. Each of these are then divided further, e.g., reliability consists of maturity, availability, fault tolerance, and recoverability.

RTT
Round-Trip Time, the time between sending a request to a remote system and getting an acknowledgement back.

SMPP
Short Message Peer-to-Peer, a communication protocol used for SMS.

SMS
Short Message Service, traditional text messaging in the GSM, 3G, 4G, and 5G networks.

SQL
Structured Query Language, the standard language for accessing relational databases.

TCP Transmission Control Protocol, providing reliable and ordered delivery of network packets over IP networks.

UCP
Universal Computer Protocol, a communication protocol used for SMS.

UCS Universal Character Set, a way to encode character codes. Common variants of this encoding are UCS-2, which uses 2 bytes per character and UCS-4, which uses 4 bytes per character.

UTF Unicode Transformation Format, a set of ways to serialize UCS values. The most commonly used variant is UTF-8, using between one and four bytes for each character. Some systems, e.g., some modern SMPP implementations, use UTF-16. This format uses two or four bytes per character, and is compatible with UCS-2 for most character codes up to 65535.

Contents

Part I

Thesis

Chapter 1

Introduction

Communication has been an important concept in computer science for a long time, even though it did not always involve networking. In earlier days the communication was mainly between the applications and what is now called the operating system kernel [64]. Over time, applications started communicating with each other, both within the same computer and across a network. This communication would sometimes require a separate software component sitting between the communication endpoints, providing protocol conversion when one or both applications could not be changed [24]. In other cases, such a component could provide a bridge between new applications and legacy databases when there was a mismatch in the data [80], e.g., whether prices are specified with or without tax, or whether locations use town names or postal codes. The names used for the software in the middle has varied, but "middleware" [64], "gateways" [24] and "mediators" [80] seem to be the most common. In this thesis, we will use "gateways".

Many gateways use a variant of the store-and-forward architecture [32] in order to isolate producers and consumers of data from each other, leading to a more resilient system than if all operations were done in lockstep. Furthermore, by storing data in the gateway for a short time before forwarding it, the incoming data packets can be merged so that the outgoing bandwidth can be utilized more effectively. The store-and-forward architecture is also useful when there is a human on at least one end, e.g., for email and instant messaging [10], as this allows the human's computer or mobile phone to temporarily be switched off. For such systems involving humans, we will use the term "messaging gateway", even though this term is sometimes used by others for application-to-application gateways.

Gateways specialized for mobile text messages are of particular interest in this thesis. Text messages, often referred to as SMS (Short Message Service), are still popular despite being a relatively old technology, as such messages can be both sent and received by all mobile phones without any additional software installed. Therefore, SMS is frequently used all over the world by companies for sending their customers meeting reminders, authentication codes, tickets, and more. In 2019, on average about 300 000 text messages were sent every second[1] Different sources claim slightly different numbers for 2022, but all are around this level.

Text messages from companies are typically not sent from mobile phones, but from applications running on computers connected to the internet. Sending an SMS directly over the internet to the network operators is surprisingly non-trivial. First, the right operator must be selected for each message. This could previously be done by just checking the first few digits of the phone number, but due to number portability this is now much more complex. Next, the operators use different communication protocols and can have very specific requirements on the traffic.

In the spirit of "*encapsulating the concept that varies*" [34], the complexity of communicating with the operators is normally contained within a gateway specifically designed to handle SMS traffic. Such a gateway, simply referred to as an SMS gateway, is often run by a category of companies known as SMS brokers. The gateways and the brokers both offer a simplification for the senders, the gateways on a technical level and the brokers on a business level. This provides added value which senders are willing to pay for, and also creates many business opportunities to provide additional services.

This thesis is centered around messaging gateways in general and SMS gateways in particular, exploring ways to make these gateways more efficient and reliable, and thereby possibly also more profitable. The efficiency is addressed in research challenges 1 and 2, focused on network round-trip times and the software architecture, respectively. Our work on these challenges led to research contribution 1 on exponential smoothing, contribution 2 on anomaly detection, and contribution 3 on architecture analysis. The reliability is then addressed in research challenge 3, on the issues that appear when using multiple servers. This resulted in a review of the state of the art and practice in contribution 4, and two data replication protocols in contributions 5 and 6.

[1] `https://visualcapitalist.com/what-happens-in-an-internet-minute-in-2019`

1.1 System Model

We define our system model as comprising one or more entities sending messages to a messaging gateway. This gateway stores the messages and sends back acknowledgements for each one. The messages get picked up from the message storage, are sent to a selected recipient, and then deleted from the storage when the acknowledgement from the recipient comes back. This matches what Petriu *et al.* [47] calls "Pipeline with buffer", but with a buffer that is persisted in some form of storage, to avoid data loss in case of a software fault. We assume that the gateway is sufficiently effective in its CPU usage, i.e., using fast and scalable data structures and algorithms, as well as a minimum of memory allocations and locks.

There are no end-to-end acknowledgements, and the senders typically cannot resend lost messages. All message flows are independent and asynchronous, and all communication to and from the gateway is carried out using standard communication protocols which cannot be modified. All remote systems are authenticated and well behaved, so there are no denial-of-service attacks or byzantine failures [55].

1.2 Thesis Goal

Previous research has shown that the existing solutions for increasing system reliability often limit the achievable throughput. Dahlin et al. [29] examined the different types of failures which can cause server unavailability. They concluded that by using a combination of techniques, e.g., data caching on the client side, routing via separate networks, and server replication, unavailability can be decreased by up to two orders of magnitude. One way of achieving this combination is to use geographically distanced servers. WanKeeper [3], by Ailijiang et al., is a service for distributed coordination of such servers. WanKeeper is based on ZooKeeper [41], extending it by using a hierarchical design, providing low latency when operations on the same client read or write the same key-value pair multiple times. In their evaluation, they reach about 100 operations per second. This is a factor of between 5.6 and 18 more than ZooKeeper in the same configuration, but still a factor of 100 less than what we aim for in this thesis. Our earlier (not published) experiments indeed confirm that existing data replication techniques typically result in a throughput several orders of magnitude lower than when using only local operations. Furthermore, the impossibility result by Didona et al. [30] says that systems which support write transactions with more than one object, will get read operations that either block, require multiple network round-trips, or return multiple values.

We see execution time predictability and stability as closely related to both reliability and throughput, as unusually long response times could indicate some sort of issue that must be addressed. For the response time distribution we had observed, we needed a solution based on collective anomalies [26], adjusted to a more detailed model than a simple exponential smoothing, even extended with a seasonal component [81] or two [75].

For data that should be identical on all servers in a distributed system, such as the message senders' credit balance, the best solutions seem to be based on Conflict-free Replicated Data Types (CRDTs) [73]. However, they often lead to excessive network and storage usage [4, 79], and we found no variant well suited for the case when the replicated data stays constant for an extended period of time, and therefore does not need to be replicated over and over.

The overall goal in this thesis is to **understand and improve both the *throughput* and the *reliability* of a messaging gateway** consistent with our system model. Primarily, this means increasing the throughput, measured as the number of processed messages per second, and the reliability, represented by the ratio of messages which would still be delivered to the correct recipient even in case of a server failure. To minimize risk and development costs, the improvements should be achieved while keeping the required changes of the existing system architecture to a minimum.

When specifying the quality requirements for a system it may be helpful to start with an existing model or taxonomy [11, 27, 35, 49, 70], perhaps even a published standard such as ISO/IEC 25010 [45]. The ISO 25010 model uses eight main characteristics: Functional Stability, Performance Efficiency, Compatibility, Usability, Reliability, Security, Maintainability, and Portability. Each one of these is then divided into a handful of related sub-characteristics.

The concrete quality requirements addressed in this thesis are listed in Table 1.1. Following the ISO 25010 taxonomy, we consider throughput and latency to be parts of Performance Efficiency, specifically the sub-characteristics Time Behaviour. Likewise, we see scalability as part of Performance Efficiency in general, and resilience as part of Reliability. Availability is already a part of Reliability in ISO 25010.

1.3 Research Challenges in Brief

We address two groups of research challenges in this thesis. In the first group, the focus is on the throughput in systems with a single node. Here we identified two challenges, RC1 and RC2. For the second group the focus shifts to multi-node messaging gateways in order to also cover reliability, resulting in

Table 1.1: The most important quality requirements (QR) in this thesis.

ISO 25010	QR	Description
Performance Efficiency	Throughput	Throughput should be high, even on moderately powerful hardware.
	Latency	Clients should get acknowledgements for sent messages without unnecessary delays.
	Scalability	It should be possible to run the messaging gateway in parallel on multiple machines for a higher total system throughput.
Reliability	Availability	Clients should be able to connect to the messaging gateway system and send messages.
	Resilience	Received and acknowledged messages should not be lost even if a limited number of servers fail or become unreachable.

RC3. The three challenges are listed below, and are described in more detail in Section 3.1.

RC1: Understand and model round-trip times and their anomalies
We observed a large variation in round-trip times for SMS traffic in several production environments, and wanted to get a better understanding of this value distribution, as well as find ways to use this understanding to reliably identify anomalies.

RC2: Identify architectural weak points and find ways to improve them
In order to possibly increase both the availability and efficiency of an existing messaging gateway, we wanted to know if its architecture could be improved, and if so, how.

RC3: Identify and resolve multi-node issues
As mentioned in Section 1.2, there is a conflict between reliability and throughput. We wanted to identify the exact reasons for this conflict for multi-node messaging systems, and find more suitable balance points between these requirements.

1.4 Thesis Contributions in Brief

The contributions of the papers included in this thesis are, briefly, as follows. They are described in more detail in Section 3.3. In short, contributions **C1**,

C2, and **C3** focus on the throughput, and **C4**, **C5**, and **C6** focus on the reliability while still not ignoring the throughput.

C1 A generalized exponential smoothing which works for any number of dimensions.

C2 An anomaly detection algorithm for collective anomalies [26] in data covering multiple orders of magnitude.

C3 An in-house variant of the Architectural Trade-off Analysis Method (ATAM) [52] for finding the architectural approaches with the largest effect on the system's quality attributes.

C4 A review of the state of the art and practice for multi-node systems.

C5 A description, implementation and analysis of a data replication protocol designed for store-and-forward systems with geographically separated nodes.

C6 A resilient and efficient method for replicating data normally updated in isolated bursts, such as the credit balances for system users.

1.5 Impact of Contributions

When our system model is implemented as an SMS gateway as shown in Figure 1.1, the message senders are typically companies, and the recipients are mobile network operators. The senders pay the SMS brokers to forward the traffic, so an SMS gateway requires a credit management module to keep track of the messages sent by each company and rejecting traffic when prepaid balances are depleted. The reliability of the message storage is business critical for the SMS brokers due to the per message cost from the operators.

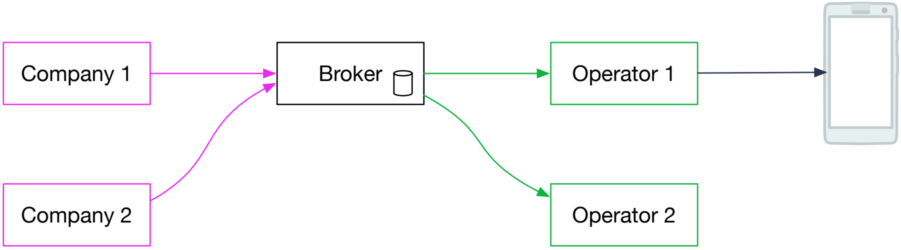


Figure 1.1: Companies sending text messages via an SMS broker to Mobile Network Operators. Originally published in Paper D.

Some SMS brokers develop their own software, while others prefer to use existing third party solutions. One of these third party products is the Enterprise Messaging Gateway (EMG) from Braxo AB. EMG is an SMS gateway matching our system model, and is used as a proof-of-concept messaging gateway and an industrial use case in this thesis. EMG handles the "soft mismatch" [24] case, as not all attributes exist or have the same values in all SMS protocols. The protocols are however similar enough for an SMS gateway to be able to provide meaningful conversions in most practical cases. Braxo gets revenues from EMG in the form of license costs paid by various SMS brokers, and from the price differences between what it charges the companies using Braxo's own EMG servers and what it pays the operators for delivering this traffic to the mobile phones.

By leveraging the throughput oriented contributions in this thesis, EMG can be made more effective in forwarding the traffic, lowering the cost for the machines it runs on, and thereby the system as a whole can generate a higher profit both for Braxo and the EMG licensees. Additionally, both SMS brokers and companies sending the messages need to be able to trust that once a message has been accepted by EMG, it will not get lost. By providing better reliability for the messages using the reliability oriented contributions in this thesis, Braxo can sell EMG licenses to more SMS brokers, and more companies will be interested in sending messages via EMG based systems.

1.6 Thesis Outline

The rest of the thesis is structured as follows.

- Chapter 2 contains further background information and elaborates on the motivation behind this thesis.
- Chapter 3 contains a summary of the research and a discussion on future work.
- Chapter 4 concludes the thesis.
- Part II contains the included papers.

Some passages of this thesis have been quoted verbatim from the author's Licentiate thesis [13].

Chapter 2

Background & Related Work

In this chapter, we describe important concepts and summarize related work relevant for this thesis. All papers in this thesis, as listed in Section 3.2, discuss SMS traffic to some degree, which motivates Section 2.1 where we describe the communication protocols used for such traffic. Log files can be very helpful in many applications in the analysis of which actions the application has taken, so the log files used in Paper A are described in Section 2.2. Our initial observations regarding the round-trip times for outgoing requests and processing times for incoming requests, which were presented in Paper N2 and analyzed in Paper A, are described in Section 2.3. Next, Section 2.4 contains a general discussion about software architecture, which is important in both Paper B, Paper C, and Paper D. Finally, Section 2.5 contains related work on the anomaly detection discussed in Paper A and the data replication discussed in Paper D and Paper E.

2.1 SMS Protocols

There are a handful of protocols used for SMS messaging, most of them originally designed for direct communication between message senders and network operators. The protocols are all similar to each other as they support almost the same set of attributes, e.g., sender phone number, recipient phone number, message body, character set, and whether a delivery receipt should be returned. The data packet containing such a set of attributes for a single request or response is called a PDU, a Protocol Data Unit.

The main differences between the protocols concern the encoding of values in a PDU. For example, UCP (Universal Computer Protocol[1]) sends all

[1] https://en.wikipedia.org/wiki/EMI_(protocol)

data as human readable text with each field separated by a "/" character, while SMPP (Short Message Peer-to-Peer[2]) sends all data as binary encoded tuples containing a field number, the data length, and the data. Over time, many SMS gateways have started to also support more general purpose protocols such as HTTP (Hypertext Transfer Protocol[3]) albeit with differences in the field names and value encodings.

The similarities between the protocols served as the basis for the creation of EMG. Thanks to SMS gateways such as EMG, otherwise incompatible SMS software products from different vendors can now easily communicate with each other.

The SMS protocols are all stateful, requiring an initial "login" operation before any messages can be sent. This means there is no need for sending authentication information with each request, and enables traffic going upstream from the mobile phones via the operators back to the companies.

Sliding windows are used to achieve a higher throughput than what would be possible if the system waited for a response after each request. Each outgoing request contains a unique transaction number, and this number must be included in the corresponding response.

The sender may want confirmation that the message was successfully delivered to the mobile phone, and would in that case request a delivery report by setting a flag in the message PDU. Such a delivery report is structured and handled in much the same way as a regular message.

2.2 SMS Gateway Log Files

Most SMS gateways can be configured to produce PDU log files, containing information about each data packet sent or received from both clients and operators. These files might then be used to view the exact network traffic, separated into the data fields used by each of the supported protocols. These PDU log files provide the data analyzed by the tool presented in Paper A.

A typical entry in a PDU log for SMPP, as it is generated by EMG, is shown below. The part "operator1,0" means the connection to "operator1", instance number 0. There can be many parallel connections to the same operator, thus the need for an instance number to distinguish them. The "trn" field is the transaction number used by the sliding window mechanism. When a response comes back with the same transaction number on the same connection, it is possible to calculate the round-trip time for that request. The "SHORT-

[2] `https://en.wikipedia.org/wiki/Short_Message_Peer-to-Peer`

[3] `https://tools.ietf.org/html/rfc7231`

MESSAGE" field contains the message body, in hexadecimal form. For a full explanation of the other fields, we refer to the publicly available SMPP specification[4].

```
2022-11-22 13:15:00.700729 INFO (operator1,0)
 Write op:0x00000004 (SUBMIT_SM)
  status:0 trn:1337 datalen:142
  SOURCEADDRTON:1 SOURCEADDRNPI:1 SOURCEADDR:464321
  DESTADDRTON:1 DESTADDRNPI:1 DESTADDR:46123456
  ESMCLASS:0 PROTOCOLID:0 PRIORITYFLAG:0
  REGISTEREDDELIVERY:1 REPLACEIFPRESENT:0
  DATACODING:3 SMDEFAULTMSGID:0 SMLENGTH:12
  SHORTMESSAGE:506844206465666656E7365
```

2.3 Round-Trip Time Distribution

The round-trip times between two servers has two main components, the time required for the data packets to travel back and forth between the servers, and the processing time on the remote machine. Briscoe *et al.* [23] have described five main groups of sources for the first component: structural delays, interaction between endpoints, delays along transmission paths, delays related to link capacities, and intra-end-host delays. Most of these are relatively constant over time between each pair of servers, which is consistent with our assumptions in Paper A where we only considered the second component, the processing time. The main exception concerns interaction between endpoints, primarily the transport initiation phase. Any variations caused by this, e.g. the TCP slow start, would be evenly distributed among different types of data packets.

In Paper N2, we looked at the distribution of the raw RTTs. The relative number of request and response pairs which completed within a certain time for one of the operators are shown in Figure 2.1. The X-axis is the time limit in μs, and the Y-axis is the relative number of pairs, both on a logarithmic scale. The larger ratio that each bullet represents, the higher up it is. The blue line shows the response times for incoming traffic to EMG, and the red line shows the response times for outgoing traffic to the operator. We see that the operator sometimes responds very quickly, shown by the left end of the red line being close to 1e+03 μs (1 ms).

[4] The most reliable way to find this document is to enter "smpp 3.4 specification" in your favourite search engine, as at the time of this writing currently there is no official owner and maintainer of this protocol. However, there are several suppliers of applications and libraries that implement SMPP who provide the specification as a service to their users.

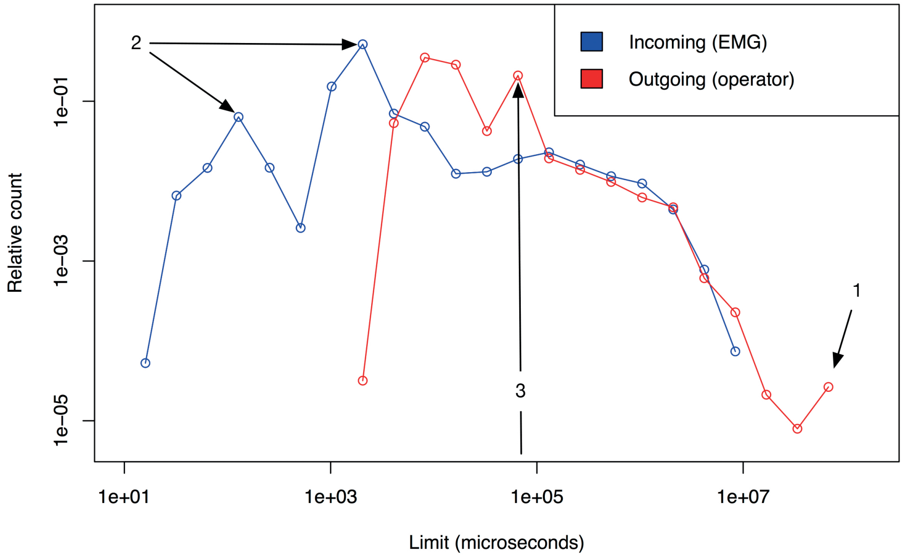


Figure 2.1: The distributions of the processing times for incoming requests to EMG, and the round-trip times for outgoing traffic to an operator.

The right end of the red line, representing messages with a RTT of more than 30 seconds, is marked with "1". If these measurements are spread evenly over a longer time span, such slow responses could actually be acceptable.

Another observation concerns the two peaks, marked with "2", on the blue line representing incoming traffic. A closer investigation revealed that the operator sends keepalive PDUs, which is a heartbeat mechanism [6] used to detect if the remote system has failed. These requests require less processing by EMG than regular messages, and this difference is reflected in these two peaks being so far apart along the X axis. From the diagram we can see that a typical keepalive message gets a response sent after around 0.1 ms (1e+02 μs), while normal messages require 5–10 ms.

Similar peaks could be seen for the outgoing traffic as well, but here the keepalive PDUs turned out to correspond to the peak to the right, in Figure 2.1 marked with "3". The operator later verified that keepalive responses were sent with a constant delay of 50 ms, which is consistent with the difference between the two peaks of the red line.

For anomaly detection to be meaningful on response times with these profiles, the data points had to be separated somehow. This separation was done in Paper A as described in Section 3.3.

2.4 Architectural Approaches

The quality attributes are often the main drivers of the system architecture [32], as the functional requirements can usually be fulfilled regardless of whether the system is a monolith on the local computer or a collection of microservices running in a cloud. The selected architecture has a much bigger effect on aspects such as response time, availability, and modifiability. The relationships between the architectural approaches used in EMG and the resulting quality attributes were explored in Paper B.

The architecture is usually one of the things that is hardest to change about a system. Making a change at the edge of a properly structured system, such as changing the SQL database from one brand to another, can sometimes be as easy as updating a configuration file. However, a change such as migrating from a monolith to microservices is an entire research field with activities both in academia [39] and the industry [40, 65]. In this field we published Paper C, on the architectural changes required to enable a messaging gateway to run in parallel on multiple servers.

We recognize that many systems can be described in multiple ways, depending on the perspective [8]. There are therefore several architectural patterns that can be used to describe the behaviour and design of a messaging gateway. They also affect which modifications are possible, and offer different advantages for the stakeholders. The patterns most relevant for this thesis are listed below, and then described in more detail.

Monolith

Enables easy development, relatively simple test and life-cycle management, and is efficient.

Store-and-Forward

Enables high availability and independent components on the system level, as the incoming and outgoing connections do not have to be active at the same time.

Publish–Subscribe

Enables high availability and independent internal components, as each connection is free to consider solely its own work.

Client–Server

Increases portability, as the gateway can be deployed anywhere in the network.

Plugins and Microservices

Provide high portability and modifiability by the application customers.

2.4.1 Monolith

Basically, a monolith has all its functionality packaged into a single executable file. This makes it easy to manage its life-cycle: either the program is running, or it is not. The main disadvantage when using a monolith is that functionality cannot be updated without restarting the entire program, which in the gateway case results in also closing all connections to both clients and operators. This connectivity issue was discussed in depth in Paper B.

2.4.2 Store-and-Forward

As mentioned, the top level architecture generally used for SMS traffic is called "store-and-forward", as the SMS gateways store each message sent to them until they can be forwarded to the right operator, and each operator stores each message until it can be forwarded to the mobile phone. This perspective is discussed to varying degrees in all papers in this thesis. A similar architecture is "batch-sequential" [32], where the focus is more on batch-wise processing of larger groups of data. Such batching was a key factor to the high throughput reached in Paper D.

Neither store-and-forward nor batch-sequential has any built-in end-to-end acknowledgement. At a lower level in the networking stack, TCP adds this acknowledgement on top of the store-and-forward based IP, but there is no corresponding mechanism for SMS. SMS indeed has delivery reports, but they are too unreliable to be used for determining whether a message must be resent. Some operators never send these back at all, forcing the SMS brokers to simply assume all messages are successfully delivered, while other operators send back positive delivery reports for all messages regardless of their final status.

2.4.3 Publish–Subscribe

The architectural style best matching how a messaging gateway might handle messages internally is called "publish–subscribe". In this architecture components called "producers" publish events on an event bus, and the "consumers" which are subscribed to matching event types are notified. The producers and consumers can thus run independently of each other. The mapping of these concepts to EMG[5] was described in Paper B, and how to replicate the event bus was the main topic of Paper D.

[5] In EMG, a *connector* is an incoming port to which clients can connect, or a set of outgoing connections to another system.

> *“The embedded NoSQL storage acts as the event bus, the connection between producers and consumers. The publisher is driven by the incoming connector the client connects to, and the consumer is driven by the outgoing connector. The events are the text messages, and the event types correspond to the names of the connectors.” – Paper B*

Having independent connectors allows clients to send a large number of messages without requiring the designated operators to always be online and able to receive the messages. Likewise, operators can return delivery reports to the SMS gateway without requiring each intended recipient to be connected.

2.4.4 Client–Server

When seen as a client–server system, a gateway acts as both a client and a server. This is shown in Figure 1.1, where a gateway running at the SMS broker acts as a server to the clients run by the companies to the left, and as a client towards the operators to the right.

2.4.5 Plugins and Microservices

Some business logic is too customer specific to capture using only configuration options. In order to provide the customers with extension points where they can add such logic themselves, many applications support plugins. This extensibility comes at the cost of lower throughput due to increased overhead as compared to making an internal function call. Making these calls to a microservice increases the independence between the application and the customer code, as the microservice can be written in any programming language and even run on a different server. However, this is also likely to incur a cost of higher latency on account of the additional network traffic required to handle each request. Both these extension types, microservices in particular, were discussed in Paper B.

2.5 Related work

2.5.1 Anomaly Detection

Chandola *et al.* [26] identified three basic anomaly types: a) “point,” when individual data points are anomalous compared to the rest of the data, b) “contextual,” when data points are anomalous in a specific context, e.g., a temperature in a specific time of the year, and c), “collective,” when a collection of data points are anomalous compared to the rest of the data. The anomaly detection used in Paper A was a combination of contextual and collective, as it was triggered by a collection of data points, but only compared to the subset of data points with similar parameter values (e.g., character encoding and PDU type).

In addition to these three types, Ibidunmoye *et al.* [42] identified the "pattern" type, for detecting changes in the shape of a series of data points. This type matches our observation discussed in Section 2.3 regarding the two distinctive peaks in the graph.

Guyon and Elisseeff gave an overview of "Variable and Feature Selection" [37, 38] from the machine learning area, used for gene selection and text classification. This could have been used in the papers A and N2 to find the attributes which had the largest effect on the round-trip time, instead of the manual method that was used. This might also have identified variations in the round-trip times caused by combinations of attributes, which was not possible using the manual method.

Similarly to what we did in Paper A, Ibidunmoye *et al.* [43] examined endless time series streams, using "tumbling windows" (sliding windows with steps > 1, and no overlap between each window), as a preliminary phase for incrementally finding means, trends and seasonalities, followed by a detection phase.

Anomaly detection in log files very often means comparing their contents with some predefined pattern or another type of state machine [7, 76]. However, this requires that such a state machine can be defined in advance, and thus appears better suited for detecting point anomalies.

2.5.2 Architecture Analysis

Within the Software Architecture field, there is a research area on the analysis of such architectures. It is clear that in this area the connection between architecture and changeability is central, as one of the first methods published on how to do an architecture analysis, the Software Architecture Analysis Method (SAAM) [51], primarily focuses on the modifiability of the evaluated system. SAAM later got extended into SAAMCS [56], focused on complex scenarios, ESAAMI [61], adding a reusable knowledge base, and SAAMER [57], adding considerations for evolution and reusability. SAAM also got extended into ATAM [8, 52], which added both a full set of quality attributes, and the two concepts *sensitivity point* and *trade-off point*. ATAM was the method we selected for the analysis in Paper B. ATAM, in turn, was then further extended into the Cost Benefit Analysis Method (CBAM) [50, 62], adding a financial dimension.

The literature reviews by Dobrica and Niemelä [31] and Ionita et al. [44] describe a few more methods, many with modifiability as a key concept. Ionita et al. also concluded that an important benefit from all these methods is "*improved communications between stakeholders*". It may therefore in some cases be more important to actually carry out an architecture analysis, than to use some particular method.

2.5.3 Data Replication

At its core, a messaging gateway is essentially a kind of message queue, an application type which exists in many variants [25, 58]. The basic idea is that some sort of message enters the system, and is delivered to one or more other systems. The most well known of these products are probably Apache Kafka [53] and RabbitMQ.[6] These products seem to fit best into publish–subscribe scenarios where each recipient subscribes to a small subset of the full data flow.

The publish–subscribe model may make it seem like a good idea to use Apache Kafka in EMG to draw on its proven throughput and data safety, but the similarities make the mapping of the quality requirements difficult. Using Kafka would also increase the complexity of deployment and management of the messaging system for the SMS brokers beyond what appears reasonable.

Worth mentioning is also the group of protocols designed for distributed agreement, primarily of an ordered sequence of events [67]. These protocols, e.g., Paxos [54] and its variants Mencius [60] and AllConcur [69], as well as Raft [66] and ZooKeeper [41], are most suitable for slow moving sequences. In the messaging context, they could be used to maintain the system configuration or the list of clients authorized to use the system, as such information is usually quite stable over time. They would be less of a fit for things such as the message queues, as using them would lead to excessive network traffic and data storage volumes. Paper C explored this in more detail, matching replication methods to the various types of data used in a messaging gateway. The replication protocols presented in D and Paper E were then designed to be more bandwidth friendly than any of these protocols.

[6] `https://www.rabbitmq.com`

Chapter 3

Research Summary

The research in this thesis is based on three main challenges, described in Section 3.1. Each challenge has been addressed in one or more of the included papers, which are described in Section 3.2. The outcome of this work is centered around six main contributions, described in Section 3.3. The overall research process and the individual research methods are then described in Section 3.4. Finally, some possibilities for future work are discussed in Section 3.5.

3.1 Challenges

According to our system model described in Section 1.1, we assume that the gateway is sufficiently effective in its CPU usage. The overall focus in this thesis is therefore on I/O related research challenges (RCs), primarily the network traffic. For RC1 and RC2 the focus is on systems with a single node, and for RC3 the focus is on systems with multiple nodes.

RC1: Understand and model round-trip times and their anomalies

The processing time required when receiving a message can be substantial, sometimes much longer than the raw network round-trip time. Some sources we have seen in production environments for this time consumption are database operations and communication with other services, e.g., for deciding to which operator a message should be sent and what the exact cost for the sender will be. These operations make the total time from sending a message to receiving its acknowledgement follow a more complex distribution function, shown in Figure 2.1 in Section 2.3, than what has been seen at lower levels in the networking stack, e.g., for ACK packets [2, 46]. Research Challenge 1 was to get a better understanding of this distribution, and find ways to identify abnormal delays.

Albeit in a different context, Underwood et al. [78] showed that the degree of variation of the network latency was highly correlated with the total performance. As the SMS protocols use windowing to limit the number of outstanding network requests, a single long round-trip prevents multiple messages from being sent during that time. It therefore seems reasonable to assume that their result holds for us as well. By identifying and thereby possibly enabling the reduction of RTT variations, resolving this challenge would get us closer to our research goal of achieving a higher throughput.

RC2: Identify architectural weak points and find ways to improve them
Before making any architectural changes, it is necessary to know the status of the system you are changing. Research Challenge 2 was therefore to identify any performance related weak points in the current architecture of EMG, and to find ways to improve its architecture.

RC3: Identify and resolve multi-node issues
Scaling a set of microservices to more nodes for increased total system performance can be relatively easy, by just moving one or more services to new nodes which can share the work load. Having the same service on multiple nodes can also provide higher resilience, as this allows the system to keep functioning during a server failure. Doing the same type of scaling or load balancing with a monolith is much more difficult, as there are typically no services to move. Also, as we saw in Section 1.2, there is often a trade-off between the increased reliability achieved by using multiple nodes and the overall system throughput, caused by the overhead from the unavoidable communication between the nodes. Research Challenge 3 was to identify these issues in more detail and find candidate solutions for our system model.

3.2 Papers

Figure 3.1 is an extension of Figure 1.1 where a second gateway with its own MySQL server and a NoSQL cluster have been added. It illustrates how each one of the included papers fits in the overall SMS gateway architecture. Each paper also addresses one or two quality attributes of a messaging gateway, listed in Table 1.1 in Section 1.2. First, Paper A focuses on the latency in the communication between senders and a gateway, and between a gateway and

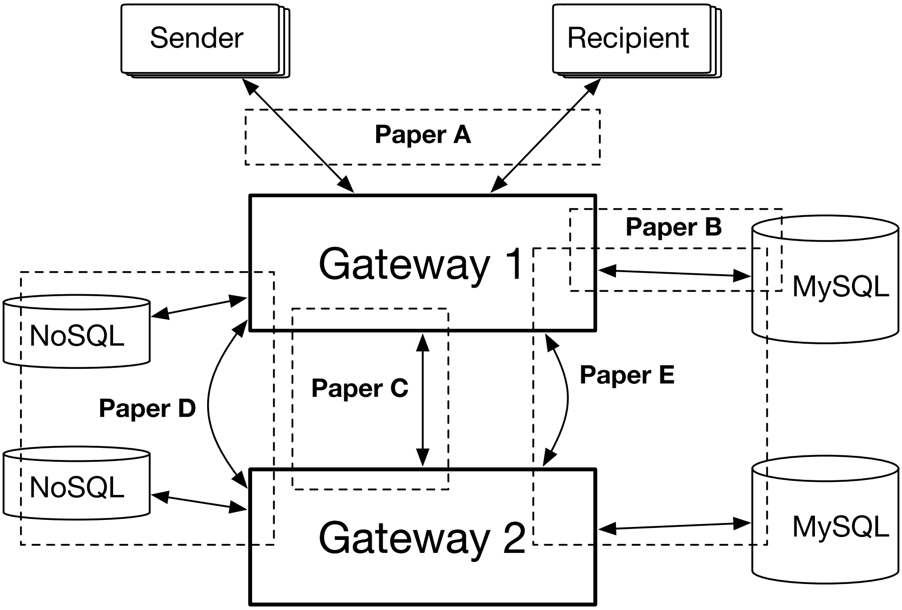


Figure 3.1: The gateway architecture and the included papers. There may be multiple senders and recipients.

recipients. Next, Paper B focuses on the gateway as such, in order to improve the availability and the communication towards the MySQL database to improve the efficiency of the system. Paper C focuses on the communication required between the gateways in a multi-node configuration to achieve good scalability and improved reliability. For the same configuration, Paper D focuses on the communication towards the NoSQL storage, and the replication of that data in order to increase the reliability. Paper E focuses on replicating the current credit balance for each client between multiple nodes.

3.2.1 Papers included in the thesis

I am the main author of all papers listed below. I also made the implementations for Paper A, Paper D, and Paper E. The co-authors all contributed with valuable discussions before and during each project, plus various additions, adjustments and clarifications of the texts. The papers have been reformatted to comply with the thesis layout, and occasional typos discovered after their publication have been corrected.

Paper A: Round-Trip Time Anomaly Detection. **Daniel Brahneborg**, Wasif Afzal, Adnan Čaušević, Daniel Sundmark, and Mats Björkman.

ACM/SPEC International Conference on Performance Engineering (ICPE), 2018 [19].

This is an extended version of Paper N2.

Paper B: A Lightweight Architecture Analysis of a Monolithic Messaging Gateway. **Daniel Brahneborg**, Wasif Afzal. IEEE International Conference on Software Architecture (ICSA), 2020 [14].

Presentation: `https://youtu.be/DHrVZeAoZoQ`

Paper C: Towards a More Reliable Store-and-forward Protocol for Mobile Text Messages. **Daniel Brahneborg**, Wasif Afzal, Adnan Čaušević, and Mats Björkman. Workshop on Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems (ApPLIED), part of the ACM Symposium on Principles of Distributed Computing (PODC), 2018 [17].

Paper D: GeoRep – Resilient Storage for Wide Area Networks. **Daniel Brahneborg**, Romaric Duvignau, Wasif Afzal, and Saad Mubeen. IEEE Access, 2022 [21].

This is an extended version of Paper N4.

Paper E: Resilient State-based CRDTs without Atomic Broadcast. **Daniel Brahneborg**, Wasif Afzal, and Saad Mubeen. International Conference on Software Technologies (ICSOFT), 2022 [20].

3.2.2 Papers not included in thesis

Additionally, I have been the author or co-author of the following papers.

Paper N1: A Pragmatic Perspective on Regression Testing Challenges. **Daniel Brahneborg**, Wasif Afzal, Adnan Čaušević. International Conference on Software Quality, Reliability & Security (QRS), 2017 [16].

Paper N2: A Black-Box Approach to Latency and Throughput Analysis. **Daniel Brahneborg**, Wasif Afzal, Adnan Čaušević. International Conference on Software Quality, Reliability & Security (QRS), 2017 [15].

This is an early version of Paper A.

Paper N3: Doctoral Symposium: Leaderless Replication and Balance Management of Unordered SMS Messages. **Daniel Brahneborg**. International Conference on Distributed and Event-based Systems (DEBS), 2019 [12].

Paper N4: Superlinear and Bandwidth Friendly Geo-replication for Store-And-Forward Systems. **Daniel Brahneborg**, Wasif Afzal, Adnan Čaušević, and Mats Björkman. International Conference on Software Technologies (ICSOFT), 2020 [18].

Winner of the Best Paper Award. It was then extended into Paper D.

3.3 Contributions

As the research challenges were resolved, we got a set of techniques for understanding and improving the throughput of a store-and-forward system, as well as for improving its reliability while reducing this throughput as little as possible. Table 3.1 maps the individual contributions to the research challenge they address, and in which paper they were described. The contributions are described in more detail next.

Table 3.1: The individual contributions, the research challenges they address, and the paper they were described in.

Contribution(s)	Challenge	Paper
C1, C2	**RC1** – Round-trip times	**A**
C3	**RC2** – Architectural aspects	**B**
C4	**RC3** – Multi-node issues	**C**
C5	**RC3** – Multi-node issues	**D**
C6	**RC3** – Multi-node issues	**E**

C1: Exponential smoothing in multiple dimensions

Our first step in addressing RC1 was to model the round-trip times based on an extended variant of exponential smoothing. Exponential smoothing is a simple technique for calculating the average of a series of values, giving more importance to values later in the series. This is done by taking a weighted sum of the previous average and the next value, and repeating that calculation for each new value. The more weight that is given to the first term, the more stable the average becomes as it responds slower to changes in the input values.

The exponential smoothing technique exists in several variants. The simplest one is basic exponential smoothing, which uses the formulas

$$\begin{aligned} s_0 &= x_0 \\ s_t &= \alpha x_t + (1 - \alpha) s_{t-1}, t > 0 \end{aligned}$$

where x_t represents each new value, α the weight between 0 and 1 for the new value, s_{t-1} the previous average, and s_t is the calculated new average. The Holt-Winters variant [81] adds a seasonal component to this, for example to handle forecasting of electricity usage which varies depending on whether it is night or day. In order to also handle the difference between summer and winter, Taylor extended this into Double Seasonal Exponential Smoothing [75].

We made the observation that SMS messages have several independent attributes, e.g., the message encoding used and whether a delivery report is requested. Each value of each such attribute may have a measurable effect on the round-trip time, so to calculate the expected RTT for a given combination of attribute values, each such combination would require its own exponential smoothing calculation. It would, however, lead to a combinatorial explosion of s_t values even though they were highly correlated with each other.

Our contribution C1 contains two parts. First, we modified the exponential smoothing to handle multi-dimensional data, by using a single s_t value with adjustment factors based on the message attributes. Second, we generalized the calculations to handle any number of such dimensions.

C2: Anomaly detection algorithm

Our second step in addressing RC1 was to create an anomaly detection algorithm designed for endless time series of round-trip times at the application level. Similarly to Agarwala et al. [1] and Tukey [77], we identify anomalies as local spikes in the input data significantly exceeding the median or mean as calculated over a longer period. As individual RTT values could vary significantly during normal operations, we focus on *collective* anomalies [26] where the median RTT over a short period differs from the median RTT over a longer period. The algorithm uses the following steps.

1. The data is normalized and some noise is removed, using the exponential smoothing described as C1 above.

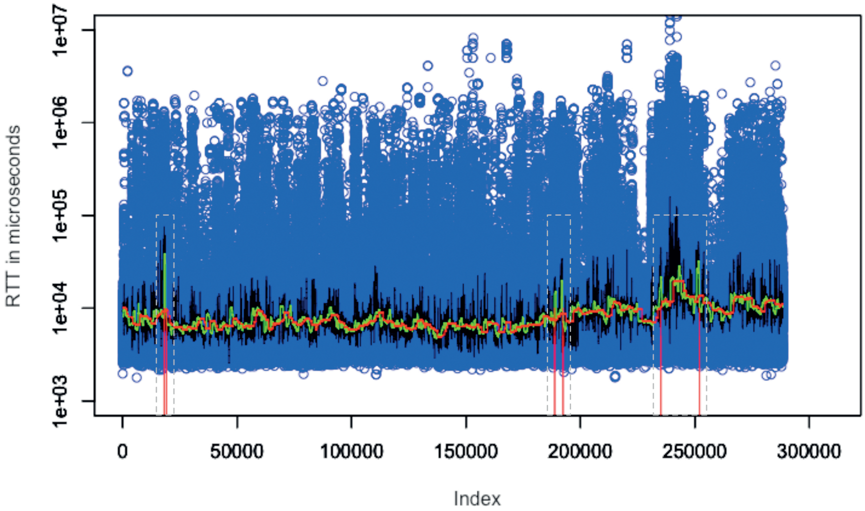


Figure 3.2: Round-trip times in blue, with three periods of unusually slow responses shown as the black and green spikes. Originally published in Paper A.

2. The Remedian [71] with a width of 5 is used to find approximate median values over several time scales. In short, this means saving the first 5 values in array A_0. When the array is full, the median of these values is appended to a second array A_1, and the next 5 values form the new A_0. This process repeats for as many levels as needed into A_2, A_3 and so on. We use the median of array A_3 as the local value and the median of array A_5 as the long term value.
3. When the local value is more than twice the long term value, the start of an anomaly is reported.
4. When the local value goes below the long term value, the end of the anomaly is reported.

Using the median of A_3 instead of individual values filters out shorter sections of slow round-trips, reducing the number of false positives. In other contexts, the size of the arrays and the selection of arrays for the short and long term values can easily be adjusted to optimize the precision. An example of the output of this algorithm is shown in Figure 3.2, for a data series with three groups of slow responses.

This anomaly detection algorithm was then implemented in a tool for analyzing existing log files. The tool revealed that one operator had response times for UCS-2 messages twice of what could be expected (for details, see Paper A), and has since then come to practical use answering multiple questions from SMS brokers regarding slow responses in their own production environments.

C3: In-house architecture analysis

In order to resolve RC2, identifying any architectural weak points in EMG, we needed a structured and proven analysis method. After comparing various potential candidates, we selected ATAM (Architectural Trade-off Analysis Method) [52]. The original version of this method describes how the various stakeholders should meet and discuss the various quality attributes, voting on the ones most important to them. The core of our contribution C3 is an extension to ATAM, describing how this voting may be replaced by simply focusing on the quality attributes related to the largest number of business drivers. This change was motivated by the observation that in many cases, each one of the business driver groups would be represented by at least one stakeholder. We therefore assumed that an attribute would have to be relevant to a large number of business driver groups in order to get a large number of stakeholder votes. We further assumed that such an attribute would indeed get a large number of votes. For the purpose of finding the most significant quality attributes, it may be just as effective to simply count lines in a diagram.

In step 5 of ATAM, the quality attributes should be described as falsifiable requirements. However, due to the wide variety in the requirements by the EMG users, this was not meaningful for us. Instead, they were described in softer terms [49], significantly reducing the amount of work required for the analysis.

Despite these significantly simplifying changes in how the attributes were described and selected for further analysis, ATAM was still able to provide us with new insights. In the mapping of the relationships between business drivers and quality requirements for an SMS gateway, as shown in Figure 3.3, the most important quality attributes are shown in **boldface**. Based on these, we concluded that extracting the credit management into a suite of microservices would have most beneficial impact for the SMS brokers, improving system availability albeit at the cost of more complex installation and maintenance procedures.

C4: Review of state of the art and practice for multi-node systems

As the first step for addressing RC3, we wanted to identify the issues that would arise when increasing the system reliability by using multiple

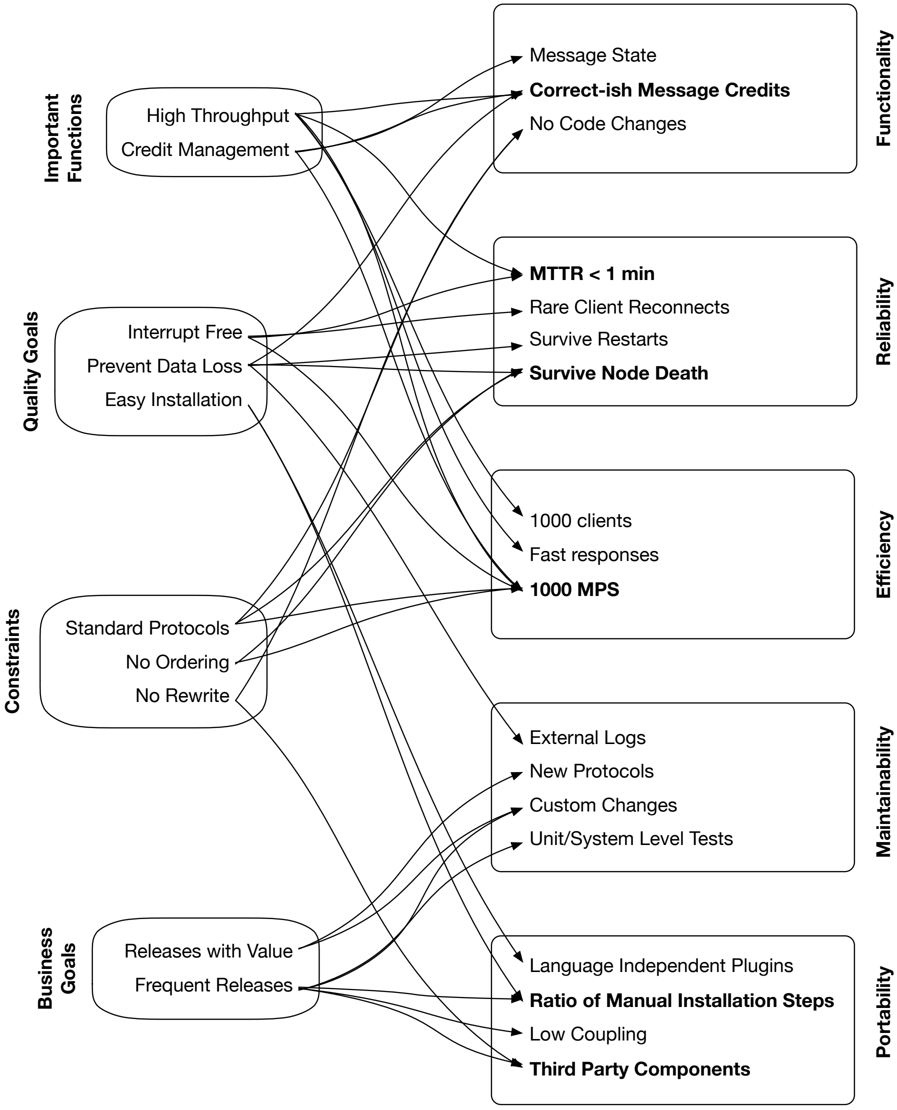


Figure 3.3: Business Drivers (left) and Quality Attributes (right) for EMG. The graph is rotated 90 degrees anti-clockwise compared to the original, which was published in Paper B.

nodes. This work led to contribution C4, which consists of the reviews of the state of the art and the state of practice for multi-node solutions, and how the different approaches relate to the requirements of a messaging gateway with an architecture similar to our system model. Typically, existing solutions for these requirements work best in local networks with high bandwidth and low latency.

The review is summarized in Table 3.2. The *System membership* requirement means keeping track of the set of nodes in the system. This information must be maintained on all nodes. The *Message storage* requirement is a fundamental part of the store-and-forward architecture. This can be solved by using a replicated database or a message queue, but existing solutions typically require a strict ordering to be maintained, which in turn consumes both CPU and network resources. We continued to work on this requirement in papers N4 and D, summarized as contribution C5 next. The *Message state* requirement is needed for billing and audit purposes. Existing solutions here, usually replicated databases or event logs, can lead to long round-trip times for the clients and limited system performance, as they use a single node for serialization of the operations. The *Message ownership* requirement is used when determining which node will forward which message. This also easily leads to a serialization node being the performance bottleneck. Finally, maintaining *Client credits* is necessary for correct billing. We identified PN-counters [73] as a possible solution for this, even though those may lead to occasional overdrafts. This requirement was further explored in Paper E, resulting in contribution C6.

Table 3.2: Considered approaches for each set of requirements, and expected new problems.

Requirement	**Approach**	**Problem**
System membership	Per node	None known
Message storage	Database	Strict ordering
	Message queue	Strict ordering
Message state	Database	Round-trip times
	Replicated log	Single node
Message ownership	Replicated log	Single node
Client credits	PN-counter	Possible overdrafts

C5: Replicated message storage

When the multi-node issues were identified in contribution C4, the next step in addressing RC3 was to find a better solution for the requirement *Message store*. The solution we developed, named GeoRep, is the core of our contribution C5, the high level description of an extensively evaluated data replication protocol tailored for store-and-forward architectures in general and messaging gateways in particular. Combined with an open sourced implementation[1], it should be possible for other researchers and engineers to modify and reimplement the protocol as needed. We also showed that in an SMS context, it is possible to achieve throughput that increases with the number of nodes in the system up to almost 3500 MPS per node, see Figure 3.4 ($24085/7 \approx 3441$). Table 3.3 shows the throughput for four different system configurations. When running within the same data-center, the throughput is similar for Paxos and GeoRep. However, when using 7 geo-separated servers, GeoRep is faster than Paxos almost by a factor of 100. This result was possible to reach as there is no relative order between each SMS, so we needed neither a central master node, nor to replicate every SMS to every node.

	Number of nodes				
	3	**4**	**5**	**6**	**7**
Local/Paxos	22827	13366	16021	13798	9343
Local/GeoRep	14880	23246	29807	32762	40437
Separated/Paxos	756	356	217	211	243
Separated/GeoRep	13253	13230	15977	21345	24085

Table 3.3: System throughput for Paxos and GeoRep, within the same data-center and geo-separated. All values are in messages per second (MPS).

C6: Replicated bursty counters

Next, we wanted to resolve the *Client credits* requirement from contribution C4 as the third step in addressing RC3. One of the ideas discussed in Paper D was the use of application level knowledge in order to find a more effective solution than what would otherwise have been possible. In Paper E, we used the same idea, but applied it to much more slowly moving data, such as the client credits which can be updated in bursts.

As mentioned, a straightforward solution in the literature is to use PN-counters [73], which consist of a set of pairs of positive and negative numbers, with one pair per node. When the value is incremented on a node, the positive number P for that node is incremented, and when

[1] `https://bitbucket.org/infoflexconnect/leaderlessreplication`

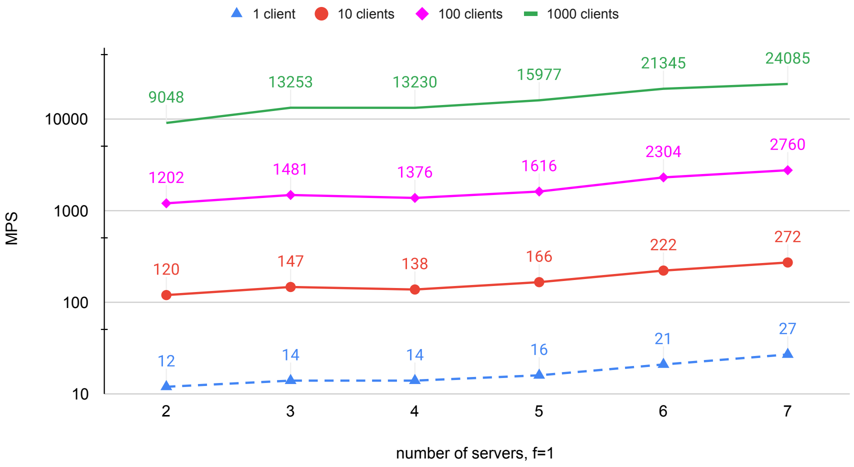


Figure 3.4: System throughput as a function of the number of active servers, running in different data-centers on multiple continents. Originally published in Paper D.

the value is decremented, the negative number N is incremented. The full set of pairs is replicated to all other nodes as needed, and each node saves the maximums of all node-specific P and N it has seen. The sum of all P minus the sum of all N gives the counter's value, which converges to the same value on all nodes.

We extended this datatype with a few attributes, e.g., at what time each pair was replicated and whether the incoming pair was identical to what a node already had. Combined, these attributes minimized the network traffic as unnecessary transmissions were eliminated, and also made the counters resilient to packet loss.

3.4 Process and Framework

Gorschek and Wnuk [36] have described a 7 step model for technology transfer between industry and academia. This model provides a roadmap for how an industry partner with a complex problem can use the knowledge and research techniques from academia to ultimately create a new solution. It also provides a suitable framework for the work in this thesis.

Figure 3.5 shows an overview of the steps in this model based on the figure in the original paper [36], adding our research challenges and contributions. Step 1 is the identification of problems, which contains all research challenges

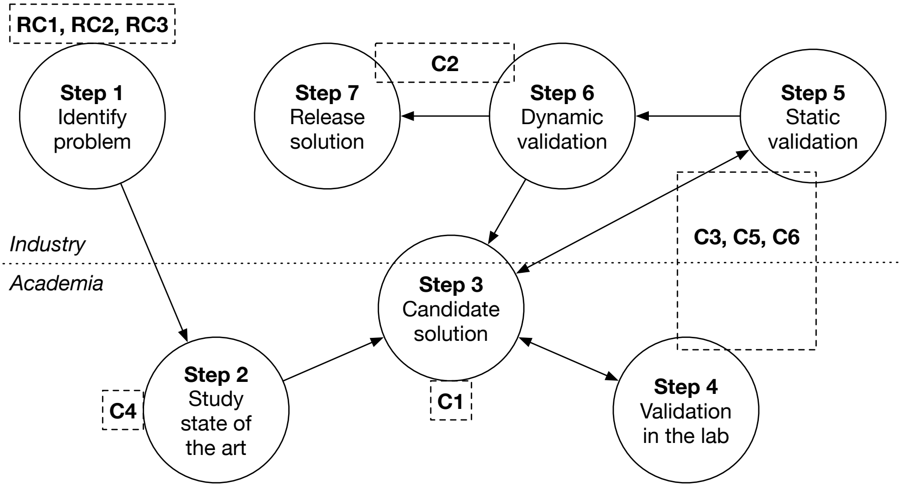


Figure 3.5: Our research challenges and contributions matching the 7 steps for technology transfer by Gorschek and Wnuk [36].

RC1, RC2, and RC3. Step 2 is for problem formulations and the study of state of the art. Here we find contribution C3. In step 3, a candidate solution is formulated, which we find in contribution C1. Step 4 is for validation in the lab, which was done in contributions C4, C5, and C6. In step 5, we go back to the industry perspective, discussing results with practitioners. This was also done in contributions C4, C5, and C6. Step 6 is for validation using pilot projects and controlled tests, as in contribution C2. Finally, in step 7 a solution is released into production. This fits the tool described in contribution C2, as it has been used repeatedly for understanding issues discovered in various production environments.

Gorschek and Wnuk [36] stress that their model is not a list of steps you follow before publishing a paper. Instead, each single step can provide sufficient new knowledge for one or more publishable research papers. Paper C is a clear example of this, as it only describes a problem and the solution space. Two of the actual solutions and their validations then come in Paper D and Paper E. Also, although it is not mentioned in Paper D, we have occasionally seen that the current GeoRep implementation has some unresolved issues on multi-core cpus and is therefore not yet ready to be used in production.

For a more detailed description of the questions in the research challenges, the approaches used to find the desired new knowledge, and the validations of the contributions, we use the framework suggested by Shaw [74]. This framework can be used both for the evaluation of existing research and for the selection of strategies for new research. As it separates the problem type, the research approach, and the result validation, it offers a more detailed and flexible framework than the more commonly used terms, e.g., “survey”, “case study”, and “experiment”.

For the selection of the type of research question, named “research setting”, Shaw suggests the classes *Feasibility*, *Characterization*, *Method/Means*, *Generalization*, and *Selection*. Of these, we used all except *Generalization*. The described research approaches are *Qualitative/descriptive model*, *Technique*, *System*, *Empirical model* and *Analytic model*. Of these, we used the first two. Shaw describes several validation techniques, grouped into *Persuasion*, *Implementation*, *Evaluation*, *Analysis*, *Experience*. We used all of them except *Analysis*.

For challenge RC1, on the nature of round-trip times, our first research setting was of the type *Characterization*, as we wanted to get a better understanding of how these values varied. To find that understanding, we used *Technique*, developing the formulas in contribution C1. In the second research setting, *Method*, we again used *Technique*, now for the development of the algorithm in contribution C2. These contributions were then validated together using *Implementation* in the form of a case study, first using existing log files from a production EMG installation and then with simulated data.

For challenge RC2, on possible architectural weak points, the research question type was *Selection*, as the goal was to find the best architectural changes. Using ATAM [8, 52], the selected approach was to generate *Descriptive models* as contribution C3. This contribution was validated using *Experience*.

For challenge RC3, on multi-node issues, we first used the question type *Feasibility* to get a better understanding of the issues that had to be resolved to achieve an efficient multi-node configuration, primarily what would be required for the network communication between the servers. The approach was again to generate a *Descriptive model*, this time as contribution C4. The validation was done using *Persuasion*.

Challenge RC3 was then addressed using *Method/Means*, to see if we could achieve higher throughput by leveraging our domain knowledge to make trade-offs and optimizations, which would not be possible in a general replication protocol. The approach was *Technique*, in the form of the replicated message storage protocol in contribution C5. This protocol was validated using a quantitative *Evaluation*. The same triple with *Method/Means*, *Technique*, and *Evaluation*, was used for C6. Table 3.4 provides an overview of the settings, approaches, and validation techniques used.

Table 3.4: The research contributions and their Shaw classifications.

Contribution	Setting	Approach	Validation
C1	Characterization	Technique	Implementation
C2	Method	Technique	
C3	Selection	Descriptive	Experience
C4	Feasibility	Descriptive	Persuasion
C5	Method	Technique	Evaluation
C6	Method	Technique	Evaluation

3.5 Future Work

The goal in this thesis, as described in Section 1.2, was to improve the quality attributes of a messaging gateway. Of the attributes described in ISO/IEC 25010 [45], particular focus has been given to Performance Efficiency and Reliability. Going forward, we have identified various ways to further improve both these and a few more attributes described in that quality model.

The ideas below are all based on concrete problems from industrial production environments. Their research settings are either *Feasibility* or *Method/ Means* [74], but the selection of a suitable research approach and result validation techniques remains open for now.

3.5.1 Performance Efficiency

For the sub-characteristics Capacity, we see work on the optimizations of the networking on a lower level, such as the use of QUIC[2] in both the data replication protocols and for the transport between SMS brokers.

For the Resource Utilization, we see some room for improvement in the windowing used by all SMS protocols. This is used, similarly as in TCP, in order to increase the throughput. However, in contrast to the automatic search for an optimal window size used in TCP, SMS gateways normally use a fixed window size, leading to sub-optimal throughput or unnecessary network queues. It would be useful to leverage existing research to find better solutions here.

3.5.2 Reliability

The reliability for individual gateway nodes can be increased by leveraging the PDU log files from production environments. On rare occasions, the com-

[2] `https://datatracker.ietf.org/wg/quic/about`

bination of a large number of events happening in a specific order leads to unexpected behaviour. As the SMS protocols are stateful and bidirectional, the events triggering the issue may consist of incoming messages as well as responses to outgoing messages. Therefore, reproducing these issues so they can be understood and resolved, is often difficult. To the best of our knowledge, there are no publicly available tools which can be used to simulate both producers and consumers, even though "replay these events" tools are common. We need a more intelligent tool which can wait for data sent by the application before sending the corresponding replies, in some cases containing field values from previously received data. We call this future tool a "pdu runner".

3.5.3 Security

The analysis implemented for Paper A is an offline solution. A useful improvement here would be to extend the tool to watch log files in real time as they are written, as the SMS broker could then be notified as soon as a problem is detected. Another interesting feature for this tool is based on the discussion in Paper D on the speed of light. If we know that the remote server is some physical distance away making no response ever being able to come back in less than for example $10\,\mathrm{ms}$, a reply with a round-trip time significantly smaller than this could indicate a man-in-the-middle attack which clearly would be worth reporting. The time spent before a remote system sends a reply may vary, but the speed of light obviously does not.

3.5.4 Reliability and Security

For the combination of Reliability and Security, we could use Fuzz Testing [59]. With this testing technique, syntactically correct data is sent to an application, but with random semantically incorrect values. For example, SMS sent as multiple parts contain both a sequence number and the total number of parts. Mulliner *et al.* [63] found, among other things, that when the first number was larger than the second, some mobile phones crashed. Fuzz testing is a way of automatically finding such cases. When implemented in a tool similar to the pdu runner discussed above, we may be able to detect previously unknown vulnerabilities, both in Braxo's EMG and in other SMS gateways.

3.5.5 Maintainability

The Reusability of the tool developed for Paper A could be increased, based on the hypothesis that the round-trip times could be expressed as some type of well-known distribution instead of just as sums. The expected RTT would then have some probability for being within a certain interval, which would make anomaly detection a bit more general, instead of using the hard coded limits currently used.

Chapter 4

Thesis Summary

4.1 Discussion

Our system model, as described in Section 1.1, together with its practical implementation described in Section 1.5, presented several constraints which prevented many of the otherwise possible solutions. First, we could only monitor the traffic and update the applications running on the gateway nodes, with access to neither message senders nor message recipients. This also prevented us from making any changes in the communication protocols used with these external parties. Next, we had to be able to handle traffic peaks of up to 1000 messages per second per node.

For RC1 in particular, addressing the round-trip times, we had to compensate for the high variation seen in normal traffic. A specific connection might have a typical RTT of just a few milliseconds, but a limited number of round-trips times of several seconds would still not be a reason for concern.

The multi-node configurations addressed by RC3 were made more difficult by the SMS use case, which also often requires delivery reports to be sent back to the sender. With the additional bookkeeping required, we needed to replicate up to 10 000 write operations per second per node, when WanKeeper [3] and similar systems could provide only around 1% of this performance. More generally, we found ourselves out of sync with state of the art in multiple ways:

1. In a world focused on distributed consistency, identifying our situation as requiring distributed *inconsistency* was non-trivial.

2. Most of the existing work on data replication [9, 22, 33, 72] address long time storage, often with causal dependencies between the stored

data tuples. Our messages are normally received, stored, forwarded, and then removed, within a second.

3. To the best of our knowledge, there are no existing storage systems which allow nodes not belonging to the majority group after a net-split to keep making progress. This unnecessarily decreases the system availability for message senders.

When making 10 000 round-trips per second, the speed of light limits the distance between the nodes to 10 km [68], which is far from enough to offer protection from outages covering larger areas. We therefore had to find another solution to RC3, ignoring unnecessary consistency requirements enforced by previous work. This was made possible as our data tuples were fully independent, even if the generality of the solution in C5 therefore is limited to such settings.

In hindsight, the calculation of adjustment factors in C1 might have been even better had it also used the Remedian [71] which is central to the anomaly detection in C2. It would however still only be useful for situations where there are just a small number of possible factors, each one with a few valid values.

The data replication in C5 could have used a stream model, where messages were accepted by one gateway node, replicated in a chain to one or more other nodes, and then forwarded to the recipient. This would also allow batching, one of the key techniques used to achieve high performance in C5, and be more inline with existing work (e.g., Chain Replication [5] and Kafka [53]), but would require a more complex system configuration.

A lesson learned from all parts of the work presented in this thesis is to not be afraid of going back to basics, questioning and discarding previous solutions as needed. Albeit this strategy was born out of necessity, as neither state of the art nor the state of practice was sufficiently efficient for us, it also forced us to analyze our problems in depth and develop new solutions.

4.2 Validity Threats

All research challenges in this thesis originated from customer requests on a specific SMS gateway (EMG), so the most significant validity threat here clearly belongs to the *external* [28, 48] category, concerning whether the results would still be valid in a more general context. We have tried to mitigate this by formulating both the purpose and the contributions of each paper in as general terms as possible. One of the constraints in Paper B, the fact that all communication with external systems had to use standard communication protocols, is from this perspective an advantage. Due to this, none of the contributions use any EMG specific communication protocols.

4.3 Conclusions

The goal behind the work presented in this thesis was to improve various selected quality attributes, primarily throughput and reliability, of messaging gateways. We formulated three specific research challenges. Research Challenge 1, RC1, was to "Understand and model round-trip times and their anomalies", motivated by us seeing the round-trip times varying by several orders of magnitude even during normal operations, and that the distribution had multiple local maxima. RC2 was to "Identify architectural weak points and find ways to improve them" in an existing messaging gateway. Any aspect of the architecture that prevents a high throughput must clearly be addressed before changing individual components. RC3 was to "Identify and resolve multi-node issues", focusing on the special set of issues that arise when extending a system to run on multiple nodes.

To better understand latency, we contribute a new technique for doing exponential smoothing in multiple dimensions (C1), and a new anomaly detection algorithm which can handle large variations in what is considered normal (C2). To better understand availability and efficiency, we contribute the description of a lightweight version of ATAM, applied to an existing messaging gateway (C3), and for scalability and maintainability a review of state of the art for multi-node systems (C4). Building on the review in C4, it became clear that existing solutions, while certainly helpful, were still not always a good fit, or needed significant adjustments to be useful in our setting. As a result, we contribute two new data replication protocols (C5 and C6), improving reliability with a significantly lesser impairment of the throughput than with existing methods.

Bibliography

[1] S. Agarwala, F. Alegre, K. Schwan, and J. Mehalingham. E2EProf: Automated end-to-end performance management for enterprise systems. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2007.

[2] J. Aikat, J. Kaur, F. D. Smith, and K. Jeffay. Variability in TCP round-trip times. In *Proceedings of the SIGCOMM Conference on Internet Measurement (IMC)*. ACM, 2003.

[3] A. Ailijiang, A. Charapko, M. Demirbas, B. O. Turkkan, and T. Kosar. Efficient distributed coordination at WAN-scale. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017.

[4] P. S. Almeida and C. Baquero. Scalable Eventually Consistent Counters over Unreliable Networks. *Distributed Computing*, 32:69–89, 2019.

[5] S. Almeida, J. Leitão, and L. Rodrigues. ChainReaction: a Causal+ Consistent Datastore based on Chain Replication. In *Proceedings of The European Professional Society on Computer Systems (EuroSys)*. ACM, 2013.

[6] P. A. Alsberg and J. D. Day. A Principle for Resilient Sharing of Distributed Resources. In *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE Computer Society Press, 1976.

[7] J. H. Andrews. Theory and Practice of Log File Analysis. Technical report, Department of Computer Science, University of Western Ontario, 1998.

[8] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice, 3rd ed.* Addison-Wesley Professional, 2013.

[9] N. Belaramani, M. Dahlin, L. Gao, A. Nayate, A. Venkataramani, P. Yalagandula, and J. Zheng. PRACTI replication. In *Proceedings of the Conference on Networked Systems Design & Implementation (NSDI)*. USENIX, 2006.

[10] A. Bennaceur, V. Issarny, R. Spalazzese, and S. Tyagi. Achieving Interoperability through Semantics-Based Technologies: The Instant Messaging Case. In *LNCS 7650*, ISWC. Springer, Berlin, Heidelberg, 2012.

[11] B. W. Boehm, J. R. Brown, and M. Lipow. Quantitative Evaluation of Software Quality. In *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE Computer Society Press, 1976.

[12] D. Brahneborg. Leaderless Replication and Balance Management of Unordered SMS Messages. In *Proceedings of the Conference on Distributed and Event-Based Systems (DEBS)*, New York, NY, USA, 2019. ACM.

[13] D. Brahneborg. *Improving the Quality Attributes of a Monolithic Messaging Gateway*. Licentiate thesis, Mälardalen University, Västerås, Sweden, 2020.

[14] D. Brahneborg and W. Afzal. A Lightweight Architecture Analysis of a Monolithic Messaging Gateway. In *Proceedings of the The International Conference on Software Architecture (ICSA)*. IEEE, 2020.

[15] D. Brahneborg, W. Afzal, and A. Čaušević. A Black-Box Approach to Latency and Throughput Analysis. In *Proceedings of the Conference on Software Quality, Reliability and Security Companion (QRS)*. IEEE, 2017.

[16] D. Brahneborg, W. Afzal, and A. Čaušević. A Pragmatic Perspective on Regression Testing Challenges. In *Proceedings of the Conference on Software Quality, Reliability and Security Companion (QRS)*. IEEE, 2017.

[17] D. Brahneborg, W. Afzal, A. Čaušević, and M. Björkman. Towards a More Reliable Store-and-forward Protocol for Mobile Text Messages. In *Proceedings of the Workshop on Advanced Tools, Programming Languages, and PLatforms for Implementing and Evaluating Algorithms for Distributed systems (PODC/ApPLIED)*, New York, NY, USA, 2018. ACM Press.

[18] D. Brahneborg, W. Afzal, A. Cauševic, and M. Björkman. Superlinear and Bandwidth Friendly Geo-replication for Store-and-forward Systems. In *Proceedings of the International Conference on Software Technologies (ICSOFT)*. INSTICC, 2020.

[19] D. Brahneborg, W. Afzal, A. Čaušević, D. Sundmark, and M. Björkman. Round-Trip Time Anomaly Detection. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering (ICPE)*. ACM, 2018.

[20] D. Brahneborg, W. Afzal, and S. Mubeen. Resilient Conflict-free Replicated Data Types without Atomic Broadcast. In *Proceedings of the International Conference on Software Technologies (ICSOFT)*. INSTICC, 2022.

[21] D. Brahneborg, R. Duvignau, W. Afzal, and S. Mubeen. GeoRep – Resilient Storage for Wide Area Networks. *IEEE Access*, 10, 2022.

[22] M. Bravo, L. Rodrigues, and P. Van Roy. Saturn: A Distributed Metadata Service for Causal Consistency. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2017.

[23] B. Briscoe, A. Brunstrom, A. Petlund, D. Hayes, D. Ros, I.-J. Tsang, S. Gjessing, G. Fairhurst, C. Griwodz, and M. Welzl. Reducing Internet Latency: A Survey of Techniques and Their Merits. *IEEE Communications Surveys & Tutorials*, 18(3):2149–2196, 2016.

[24] K. Calvert and S. Lam. Formal methods for protocol conversion. *IEEE Journal on Selected Areas in Communications*, 8(1):127–142, 1990.

[25] S. Celar, E. Mudnic, and Z. Seremet. State-Of-The-Art of Messaging for Distributed Computing Systems. In *Proceedings of the DAAAM International Symposium*, Vienna, Austria, 2016.

[26] V. Chandola, A. Banerjee, and V. Kumar. Anomaly Detection: A Survey. *ACM Computing Surveys*, 41(3):58, 2009.

[27] L. Chung and J. C. S. do Prado Leite. On Non-Functional Requirements in Software Engineering. In *Lecture Notes in Computer Science*, volume 5600 LNCS, pages 363–379. Springer Berlin Heidelberg, 2009.

[28] T. D. Cook and D. T. Campbell. *Quasi-experimentation: Design and Analysis for Field Settings*. Rand McNally, Chicago, 1979.

[29] M. Dahlin, B. B. V. Chandra, L. Gao, and A. Nayate. End-to-end WAN Service Availability. *IEEE/ACM Transactions on Networking*, 11(2):300–313, 2003.

[30] D. Didona, P. Fatourou, R. Guerraoui, J. Wang, and W. Zwaenepoel. Distributed Transactional Systems Cannot Be Fast. In *Proceedings of the Symposium on Parallelism in Algorithms and Architectures (SPAA)*, New York, NY, USA, 2019. ACM Press.

[31] L. Dobrica and E. Niemelá. A Survey on Software Architecture Analysis Methods. *IEEE Transactions on Software Engineering*, 28(7):638–653, 2002.

[32] G. Fairbanks. *Just enough software architecture: a risk-driven approach*. Marshall & Brainerd, 2010.

[33] P. Fouto, J. Leitão, and N. Preguiça. Practical and Fast Causal Consistent Partial Geo-replication. In *Proceedings of the International Symposium on Network Computing and Applications (NCA)*. IEEE, 2018.

[34] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[35] M. Glinz. On Non-Functional Requirements. In *Proceedings of the International Requirements Engineering Conference (RE)*. IEEE, 2007.

[36] T. Gorschek and K. Wnuk. Third Generation Industrial Co-production in Software Engineering. *Contemporary Empirical Methods in Software Engineering*, 2020.

[37] I. Guyon and A. Elisseeff. An Introduction to Variable and Feature Selection. *Journal of Machine Learning Research (JMLR)*, 3(3):1157–1182, 2003.

[38] I. Guyon and A. Elisseeff. An Introduction to Feature Extraction. In *Feature Extraction, Studies in Fuzziness and Soft Computing*, volume 207, pages 1–25. Springer Berlin Heidelberg, 2006.

[39] S. Hassan and R. Bahsoon. Microservices and Their Design Trade-offs: A Self-Adaptive Roadmap. In *Proceedings of the International Conference on Services Computing (SCC)*. IEEE, 2016.

[40] G. Hohpe. Enterprise Integration Patterns. `https://www.enterpriseintegrationpatterns.com`, 2020 (accessed 2021-08-06).

[41] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*. USENIX Association, 2010.

[42] O. Ibidunmoye, F. Hernández-Rodriguez, and E. Elmroth. Performance Anomaly Detection and Bottleneck Identification. *ACM Computing Surveys*, 48(1), 2015.

[43] O. Ibidunmoye, A.-r. Rezaie, and E. Elmroth. Adaptive Anomaly Detection in Performance Metric Streams. *IEEE Transactions on Network and Service Management*, 15(1):217–231, 2018.

[44] M. T. Ionita, D. K. Hammer, and H. Obbink. Scenario-based software architecture evaluation methods: An overview. In *Proceedings of the Workshop on Methods and Techniques for Software Architecture Review and Assessment, part of International Conference on Software Engineering (ICSE)*, 2002.

[45] ISO/IEC. ISO/IEC 25010. `https://iso25000.com/index.php/en/iso-25000-standards/iso-25010`, 2020. Accessed 2020-06-07.

[46] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Inferring TCP Connection Characteristics Through Passive Measurements. In *IEEE INFOCOM*, volume 3, pages 1582–1592. IEEE, 2004.

[47] A. Jalnapurkar, C. Shousha, and D. Petriu. Architecture-Based Performance Analysis Applied to a Telecommunication System. *IEEE Transactions on Software Engineering*, 26(11):1049–1065, 2000.

[48] A. Jedlitschka, M. Ciolkowski, and D. Pfahl. Reporting experiments in software engineering. In *Guide to advanced empirical software engineering*. Springer, London, 2008.

[49] I. J. Jureta, S. Faulkner, and P. Y. Schobbens. A More Expressive Softgoal Conceptualization for Quality Requirements Analysis. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 4215 LNCS(May):281–295, 2006.

[50] R. Kazman, J. Asundi, and M. Klein. Quantifying the costs and benefits of architectural decisions. In *Proceedings of the The International Conference on Software Engineering (ICSE)*. IEEE, 2001.

[51] R. Kazman, L. Bass, G. Abowd, and M. Webb. SAAM: A Method for Analyzing the Properties of Software Architectures. In *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, 1994.

[52] R. Kazman, M. Klein, and P. Clements. Method for Architecture Evaluation. Technical report, Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, 2000.

[53] J. Kreps, N. Narkhede, and J. Rao. Kafka: a Distributed Messaging System for Log Processing. In *Proceedings of the SIGMOD Workshop on Networking Meets Databases (NetDB)*, Athens, Greece, 2011.

[54] L. Lamport. The Part-time Parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.

[55] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3), 1982.

[56] N. Lassing, D. Rijsenbrij, and H. van Vliet. On software architecture analysis of flexibility, complexity of changes: Size isn't everything. In *Proceedings of the The Nordic Software Architecture Workshop (NOSA)*, 1999.

[57] C.-H. Lung, S. Bot, K. Kalaichelvan, and R. Kazman. An Approach to Software Architecture Analysis for Evolution and Reusability. In *Proceedings of the The Conference of the Centre for Advanced Studies on Collaborative Research*, 1997.

[58] L. Magnoni. Modern Messaging for Distributed Systems. *Journal of Physics: Conference Series*, 608:1–8, 2015.

[59] V. J. Manes, H. Han, C. Han, S. K. Cha, M. Egele, E. J. Schwartz, and M. Woo. Fuzzing: Art, science, and engineering. *arXiv preprint arXiv:1812.00140*, 2018.

[60] Y. Mao, F. P. Junqueira, and K. Marzullo. Mencius: Building Efficient Replicated State Machines for WANs. In *Proceedings of the USENIX Conference Operating System Design and Implementation (OSDI)*, 2008.

[61] G. Molter. Integrating SAAM in Domain-centric and Reuse-based Development Processes. In *Proceedings of the The Nordic Workshop on Software Architecture (NOSA)*, 1999.

[62] M. Moore, R. Kazman, M. Klein, and J. Asundi. Quantifying the Value of Architecture Design Decisions: Lessons from the Field. In *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE, 2003.

[63] C. Mulliner, N. Golde, and J.-P. Seifert. SMS of Death: From Analyzing to Attacking Mobile Phones on a Large Scale. In *Proceedings of the USENIX Security Symposium*, volume 168. San Francisco, CA, 2011.

[64] P. Naur and B. Randell. Software Engineering: Report of a Conference Sponsored by the NATO Science Committee, Garmisch, Germany. Technical report, Scientific Affairs Division, NATO, Belgium, 1969.

[65] S. Newman. *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith.* O'Reilly Media, 2019.

[66] D. Ongaro. *Consensus: Bridging Theory And Practice.* PhD thesis, Stanford University, 2014.

[67] C. Papadimitriou. Serializability of Concurrent Database Updates. Technical report, Naval Research Laboratory, Cambridge, Massachusetts, 1979.

[68] R. Percacci and A. Vespignani. Scale-free behavior of the Internet global performance. *European Physical Journal B*, 32(4):411–414, 2003.

[69] M. Poke, T. Hoefler, and C. W. Glass. AllConcur: Leaderless Concurrent Atomic Broadcast Marius. In *Proceedings of the International Symposium on High-Performance Parallel and Distributed Computing (HPDC).* ACM, 2017.

[70] G. C. Roman. Taxonomy of Current Issues in Requirements Engineering. *Computer*, 18(4), 1985.

[71] P. J. Rousseeuw and G. W. Bassett. The Remedian: A Robust Averaging Method for Large Data Sets. *Journal of the American Statistical Association*, 85(409):97–104, 1990.

[72] N. Schiper, P. Sutra, and F. Pedone. P-store: Genuine Partial Replication in Wide Area Networks. In *Proceedings of the Symposium on Reliable Distributed Systems (SRDS).* IEEE, 2010.

[73] M. Shapiro, N. Pregui, C. Baquero, and M. Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types. Technical Report RR-7506, Inria – Centre Paris-Rocquencourt, 2011.

[74] M. Shaw. The Coming-of-Age of Software Architecture Research. In *Proceedings of the International Conference on Software Engineering (ICSE).* IEEE Computing Society, 2001.

[75] J. W. Taylor. Short-Term Electricity Demand Forecasting Using Double Seasonal Exponential Smoothing. *Journal of Operational Research Society*, 54(8):799–805, 2003.

[76] D. Tu, R. Chen, Z. Du, and Y. Liu. A Method of Log File Analysis for Test Oracle. In *Proceedings of the Conference on Scalable Computing and Communications*. ACM, 2009.

[77] J. W. Tukey. *Exploratory Data Analysis*. Reading, MA, 1977.

[78] R. Underwood, J. Anderson, and A. Apon. Measuring Network Latency Variation Impacts to High Performance Computing Application Performance. In *Proceedings of the ACM/SPEC International Conference on Performance Engineering (ICPE)*, New York, NY, USA, 2018. Association for Computing Machinery.

[79] P. Urbán, X. Défago, and A. Schiper. Contention-Aware Metrics for Distributed Algorithms: Comparison of Atomic Broadcast Algorithms. In *Proceedings of the International Conference on Computer Communications and Networks (IC3N)*. IEEE, 2000.

[80] G. Wiederhold. Mediators in the Architecture of Future Information Systems. *Computer*, 25(3):38–49, 1992.

[81] P. R. Winters. Forecasting Sales by Exponentially Weighted Moving Averages. *Management Science*, 6(3):324–342, 1960.

Part II

Included Papers

Paper A. Round-Trip Time Anomaly Detection

Round-Trip Time Anomaly Detection. Daniel Brahneborg, Wasif Afzal, Adnan Čaušević, Daniel Sundmark, Mats Björkman.

In *Proceedings of the ACM/SPEC International Conference on Performance Engineering (ICPE)*. ACM, 2018.

Abstract

Mobile text messages (SMS) are sometimes used for authentication, which requires short and reliable delivery times. The observed round-trip times when sending an SMS message provide valuable information on the quality of the connection.

In this industry paper, we propose a method for detecting round-trip time anomalies, where the exact distribution is unknown, the variance is several orders of magnitude, and there are lots of shorter spikes that should be ignored. In particular, we show that using an adaption of Double Seasonal Exponential Smoothing to reduce the content dependent variations, followed by the Remedian to find short-term and long-term medians, successfully identifies larger groups of outliers. As training data for our method we use log files from a live SMS gateway. In order to verify the effectiveness of our approach, we utilize simulated data. Our contributions are a description on how to isolate content dependent variations, and the sequence of steps to find significant anomalies in big data.

5.1 Introduction

Measuring and monitoring round-trip times (RTTs) of data packets in a networked environment is important for at least two reasons: (1) to maintain the negotiated service levels of quality and (2) to minimize operational costs. To better understand the importance of this monitoring, let us consider a scenario where a person wants to login to an Internet bank.

1. The customer goes to the bank website and enters a personal, unique identification number.

2. The bank finds this information in its customer database, and sends a verification code as an SMS message to the registered mobile phone for this customer. In most cases, the SMS arrives to the mobile phone well within the negotiated, stipulated time, but in some anomaly situations, the message could be delayed for a considerable amount of time.

3. The customer enters the verification code, completing the login procedure.

There are many mobile network operators, and as the bank does not want to maintain connections and agreements with all of them, this service is outsourced to one or more "SMS brokers". SMS brokers (SBs) manage the SMS traffic between their customers and the network operators. For increased reliability, the bank connects to two SMS brokers (Figure 5.1). Assuming the cost for sending messages via SB1 is lower than via SB2, by default the bank sends all messages via SB1. If the connection to SB1 is lost for some reason, the bank quickly switches to SB2 in order to avoid delays in the SMS deliveries. In the worst case it could take hours for the connection to SB1 to be fully functional again, so without this switch, the problem with SB1 would result in dissatisfied customers for the bank as the verification codes would stay in the outgoing queue in the bank's SMS Gateway. The sooner the switch to SB2 can be made, the smaller the delay seen by the bank's customers. Using SB2 all the time would not improve the situation, as SB2 could also become unreachable for any number of reasons, e.g. broken hardware, or problems at their Internet service provider.

The bank customer is the only one who knows the exact delivery time, and that is just for their own message. In order to get an overall view, from this point on we will use a simpler measurement: the RTT between the bank and the SMS broker.

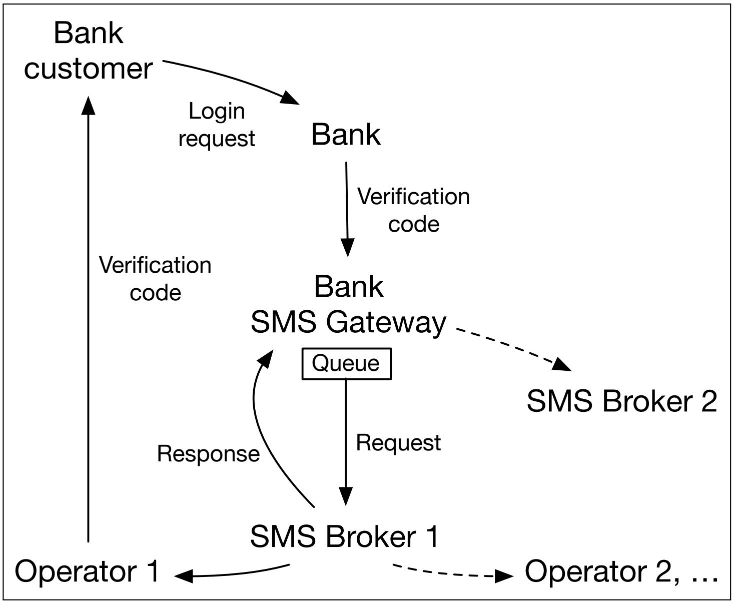


Figure 5.1: An example scenario emphasizing the importance of measuring and monitoring round-trip times: the bank customer, the bank, two SMS brokers and some operators.

In this paper we focus on detecting violations that fall somewhere between a few individual messages being slightly delayed and a fully broken connection. Let us assume the operator normally has two servers handling SMS traffic, and one of them temporarily breaks. With incoming throughput to the SMS broker being constant and outgoing throughput being halved, a queue of messages may form. To prevent this queue from growing without bounds, the SMS broker can throttle incoming traffic by delaying its responses. Clients must implement proper windowing, so these delays will cause them to delay their future requests.

The SMS system behaves much like a train of cars, in that we can draw conclusions on the situation further ahead by observing the car in front of us. If the car slows down, we can assume there is a problem with the traffic in general. Provided the slow speed persists, we might decide to choose an alternate route. Similarly, the RTT towards the SMS broker provides the client (the bank) with valuable feedback on the effective throughput of the entire chain of SMS brokers and operators.

This paper addresses the situation when the absolute values of the delivery time are not known. We know from earlier results [4] that the RTT has very few anomalies, but when they happen, we want to know as soon as possible. We have seen that there are several shorter spikes in these RTTs, so our research objective is to develop a method of automatically finding longer periods of outliers in RTTs while ignoring these short uninteresting spikes. In particular, we examine the variation of the RTTs in a production system of an SMS broker between their own system and several external operators.

Section 5.2 describes the context in more detail and Section 5.3 describes related work for RTT measurement and anomaly detection. Our approach is described in Section 5.4. We then describe our case study in Section 5.5, and the results in Section 5.6. Section 5.7 discusses these results, and the paper ends with conclusions and future work in Section 5.8.

5.2 Background and Terminology

Figure 5.2 shows the simple base scenario of the network traffic as seen from the SMS Gateway software used by the SMS brokers. The filled arrows represent SMS messages, the unfilled arrows are responses, and A, B etc. are points in time. The SMS Gateway only knows about the times B, C, E and H. The arrow from J to K is dashed, as we do not know when this event occurs. The difference between B and C shows the processing time required for an incoming message, while the difference between E and H shows the full RTT to the operator. We will examine both these differences, as anomalies between B and C reveal problems in the local environment and anomalies between E and H reveal problems in the network or with the remote node. The difference between C and E is how long the message sits in the outgoing queue, waiting to be sent. From the bank customer's point of view, the login request starts at some point before A, and the verification code arrives to the phone at K. The delivery of the message to the mobile phone and the response sent back to the SMS Gateway happens in parallel, so the relationship between H and K is undefined.

In many cases, monitoring of response times is required as a way of making sure the system works as expected. According to our industrial experience, one of two methods are commonly used for this monitoring. 1) Visualize selected measurements on a display, which is simple to implement but requires a human to look at the display. This is easily forgotten if anomalies are rare. 2) Utilize tools based on Simple Network Management Protocol (SNMP), reporting detected anomalies without requiring human interaction. A drawback is that those checks are usually trivial with static tolerance levels, e.g. whether a single RTT is longer than 1 second or whether the processing queue contains more than 1000 elements.

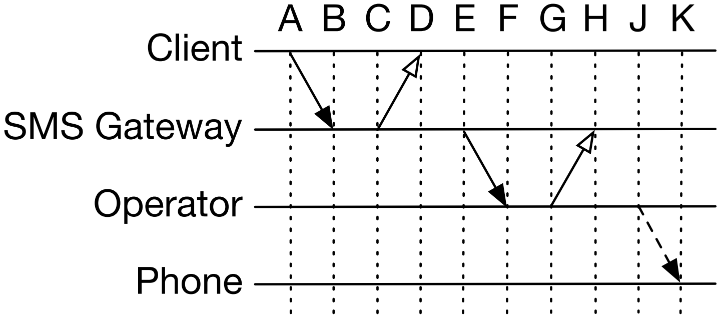


Figure 5.2: The network traffic between a client, an SMS Gateway, an operator and a mobile phone.

Once an anomaly has been detected, some fault localization technique [22] should be applied to find the root cause of the problem. This is however outside the scope of this paper.

While SMS brokers reduce the number of accounts needed, the combined network traffic becomes more difficult to analyze. Some brokers specialize in operators in a particular region, decreasing the number of accounts but increasing the number of intermediate nodes. Broker handling of messages varies, e.g. they may store the messages on disk for safety, or wait for acknowledgment from the next node before responding back to the previous. These factors incur variability in response times, even between the same nodes. We assume that if a node uses a server cluster, all these servers are homogeneous, giving consistent RTTs.

5.2.1 Terminology

We will now define the concepts discussed in this paper:

Node: Common term for clients, operators and SMS brokers.

Downstream/Upstream: Downstream is as ordered in Figure 5.2, i.e. client to SMS broker to operator to mobile phone. Upstream is, obviously, the reverse direction.

Request: A data packet containing an SMS message, including the sender, recipient and message body, or a delivery report.

Response: Acknowledgement of a received request.

PDU: Protocol Data Unit, refers to both requests and responses.

Delivery report: A data packet sent as confirmation of successful message delivery to the recipient or rejection by a node.

Round-trip time (RTT):! For outgoing traffic, RTT is the interval between when the request is sent and the response comes back. For incoming traffic, RTT is the interval after receiving a request until the response is sent. In Figure 5.2, these are the intervals from E to H and from B to C respectively.

Throughput: The number of messages received and forwarded by a node, per some specified time unit.

Window size: The number of requests the client sends before waiting for a response.

Outlier: A single RTT measurement significantly higher than usual for a specific connection. Responses arriving earlier than usual is both very rare and typically not a problem.

Anomaly: A larger cluster of outliers. This is defined in more detail in Section 5.4.4.

5.3 Related work

Earlier studies have focused on either RTT measurement or anomaly detection, so we will describe these groups of papers separately.

5.3.1 RTT measurement

For RTT measurement, existing work can be structured according to what protocol they analyze. A relatively common layer for RTT measurements is TCP, as it is used for many applications and therefore enables analysis of large amounts of data. Here we find an examination of several different TCP implementations [18], and a description of the experiences using the tool Tstat [16].

TCP includes an ACK packet which, similarly to our response PDUs, provides an easy way to calculate the RTT. The RTT can then be either approximated using just the SYN/ACK pair used to initiate the connection [12] or more correctly using also the data packets and their responses [26]. Martin et al. [15] took this further by using the minimum and average values of the RTT for both the SYN and data packets to separate the physical latency from the server side processing time. The packet-pair strategy was then generalized for raw IP traffic [27].

At the application layer, which is most similar to our work, we have studies on HTTP traffic by Mosberger and Jin [17] using their tool `httperf`, and Halepovic et al. [9] who examined the RTTs from mobile clients to web servers. The throughput values given by `httperf` had an average close to the maximum, which corresponds to an average RTT being close to the minimum.

In some cases, the minimum and maximum RTT values are the most interesting [8], in which case there is no need to examine the distribution in more detail. Papers that have analyzed the data deeper, have found variances in RTT for TCP traffic between 1 millisecond and 200 seconds [11, 2]. In an analysis for Controller Area Networks, the type of network used in real-time environments, the data had a good fit with the Gamma distribution [28]. Taken together, most papers that have examined the distribution of RTT values, explicitly or implicitly describe it as exponential in some way. This is consistent with our findings.

5.3.2 Anomaly detection

Shanbhag and Wolf suggest using multiple anomaly detection algorithms in parallel [21], and using the combined result as the trigger. Even though we do not use multiple algorithms, we use all relevant data fields in the PDU to calculate the expected values with as much precision as possible.

E2EProf, as described by Agarwala et al. [1], is similar to our approach as it also uses time-series analysis, of which exponential smoothing is one of the methods, to analyze the performance of each subsystem of an application. They define a “spike” as a local maximum, exceeding a threshold of the mean plus three times the standard deviation. For testing, they used `httperf`.

Bayesian Principal Anomaly Detection (BPAD) warns for individual outliers [10], and because these occur too frequently, it does not suit our context.

Between the years 2000 and 2010, there were several papers [19, 14, 13] on using Principal Component Analysis (PCA) for anomaly detection. Even though the method worked fine, it was difficult to find the right sensitivity [5].

Wang et al. [25] stress that anomaly detection methods must in some circumstances be “lightweight”, both in terms of the number of metrics they require to run (the volume of data continuously captured and used), and in terms of their runtime complexity. They suggest smoothing the data, just as we use exponential smoothing (Section 5.4.2), and detect anomalies using the Tukey method based on “fences” and “hinges” [24]. This method splits the data into quartiles separated at Q1, Q2 and Q3, and classifies anomalies in multiples of the difference between Q1 and Q3. While different from our method, it also uses the median instead of the mean.

5.4 Approach

In order to understand the RTT values, we first calculated the mean and standard deviation of a few collections of RTTs (Section 5.4.1). We then used exponential smoothing to get a mean value that gave higher importance to newer RTTs (Section 5.4.2). Some parts of the variance turned out to be related to specific aspects of the message data, so the exponential smoothing was further refined to isolate these as adjustment factors (Section 5.4.3). Finally we calculated the median of smaller and larger groups of RTTs as a way of identifying outlier clusters (Section 5.4.4).

5.4.1 Mean and standard deviation

As mentioned in Section 5.2.1, the time spent by a node processing a request can vary significantly, so the RTT varies from fractions of a millisecond to multiple seconds. Calculating the mean from such data does not give meaningful results.

The exact distribution of the RTTs is not known to us. However, earlier work shows that it resembles a log-normal distribution, so we calculate the mean and variance of the logarithms of the RTTs.

For efficiency, we use formulas based on those described by Finch [7]. The formula used for the incrementally calculated mean is shown in Equation 5.1. Here, x_n is the new value, and n is the number of values so far. We use Equation 5.2 to get the variance S_n, and Equation 5.3 for the standard deviation σ_n.

$$\mu_n = \mu_{n-1} + \frac{1}{n}(\ln x_n - \mu_{n-1}) \tag{5.1}$$

$$S_n = S_{n-1} + (\ln x_n - \mu_{n-1})(\ln x_n - \mu_n) \tag{5.2}$$

$$\sigma_n = \sqrt{S_n/n} \tag{5.3}$$

5.4.2 Exponential smoothing

Over time, the effect of new values added to Equation 5.1 shown in Section 5.4.1 will diminish. By instead using exponential smoothing, we are able to analyze an endless series of data.

We calculate the expected value E_n using the well-known Equation 5.4, where n is the number of observations, and V_n is the nth value. Or rather, V_n is the logarithm of the measured RTT, and E_n is the logarithm of the expected value. The new value is the sum of two terms based on the current observation and on the previously expected value, respectively. The constant α is used to select the scaling factor between them, where a lower value of α gives a more stable E_n, as the effect from individual values of V_n is smaller. We set E_1 to V_1.

$$E_n = \alpha V_n + (1 - \alpha)E_{n-1},\ n > 1 \tag{5.4}$$

5.4.3 Adjustment factors

As the traffic between SMS brokers uses Internet, network related RTTs can vary both by time of day and day of week. While grouping the data by hour gives a lower variance and therefore improved anomaly detection, it also gives less data in each group, resulting in reduced stability. Moreover, it disregards the similarities of RTTs during consecutive hours.

Communication protocols for SMS consist of fields with key-value pairs which specify how the SMS should be handled, so we assume their values might affect the RTT. To minimize the variance, each unique combination of fields should be analyzed separately. This strategy leads to a combinatorial explosion, and requires large amounts of data for satisfactory stability of E_n. In the financial domain Double Seasonal Exponential Smoothing [23] is sometimes used, basing the result on time values, e.g. day of month and month of year. The idea is to get a single average value for the entire dataset, with a small number of adjustment factors. Similar approaches have also been used in network contexts [6]. We use a variation of this method, but with field values instead of time values.

We need one adjustment factor per field value, and use the syntax F^v_n for the nth value of the adjustment factor for field value v. The value of F^v_0 is set to 0, representing the case when the RTT is identical for all values. The adjustment factor can be either additive or multiplicative, and because of the exponential nature of the RTT distribution, multiplicative adjustments seem to make the most sense. However, as the values of E_n and V_n are logarithms, the actual adjustment needs to use addition. The calculation of the effect from a specific field value is shown in Equation 5.5. We want the expected value E_n to be free from these variations, so Equation 5.4 is modified to instead use the adjusted value of V_n, as shown in Equation 5.6.

$$F^v_n = \alpha(V_n - E_n) + (1 - \alpha)F^v_{n-1} \tag{5.5}$$

$$E_n = \alpha(V_n - F^v_{n-1}) + (1 - \alpha)E_{n-1} \tag{5.6}$$

For the more general case, we see the difference between the expected value E_n and the measured value V_n as the sum of all adjustment factors for all fields. We can then update the adjustment factors using the same exponential smoothing as in Equation 5.4. This is shown in detail in Algorithm 1, lines 7 to 13. For simplicity we use the same scaling factor α as for the expected value in Equation 5.4, but it is possible to use different scaling factors for each adjustment factor.

5.4.4 Medians

Even with α as low as 0.0001, the wide range of values in the input data renders E_n too unstable to be useful in detecting anomalies. A more reliable reference point is given by the median, in our case calculated using the Remedian [20] method. The algorithm is simple but effective, using k arrays A_i, each of length b.

1. Store the first b values in A_0, where typically $b < 10$.
2. Calculate the median of A_0 and append the result to A_1.
3. Repeat steps 1 and 2 until A_1 contains b values. Calculate the median of them, and append the result to A_2.
4. Repeat the previous steps up to A_k for all i less than some k, appending the median of A_{i-1} to A_i.

The median of A_k is now an estimate of the median of the full series of values. The number of operations required to find the median of b values is fixed for each b, giving an execution time complexity of $O(n)$ for n values. We can think of it as a software version of multiple connected Geneva drives [3].

The value we append to A_0 is E_n, the most recent measurement with all adjustment factors removed. Using arrays with $b = 5$ values each achieves a good balance between stability, which requires more values in each array, and sensitivity, which requires fewer values. This way A_0 has the median of the most recent 5 values, A_1 of 25 values, A_2 of 125, etc.

We can now define an anomaly as a cluster of outlier measurements that increase the median of A_3 above twice the median of A_5. To avoid repetitive notifications, each notification suspends further ones until the median of A_3 goes below the median of A_5. A period of large values that is long enough to affect the median of A_3 this way occurs sufficiently seldom, as shown in Section 5.6.3.

5.4.5 Summary

Algorithm 1 combines the steps described earlier in this section. The algorithm is implemented as an extension to our existing tool called ELFA (EMG Log File Analyser – initially introduced in [4]). The method described here has several benefits.

1. All calculations are done in constant time, depending only on the number of adjustment factors. This is necessary as we need to be able to handle continuous traffic with up to 1000 measurements per second.
2. The sensitivity is easily adjusted, even online.
3. It is independent of the frequency of values.
4. The expected value is calculated from all observations, not just from an artificial subset.
5. Adjustment factors can be added and removed online as needed.
6. For each connection we need to persist only the base value E_n and the non-zero adjustment factors F_n^v to be able to resume a paused analysis.
7. It is self-adapting, using the most recent RTT values for each individual connection as the basis for detecting outliers.

5.5 Case Study Design

To evaluate our approach for detecting anomalies in the RTTs, we undertook an industrial case study. Specifically, we wanted to investigate and exemplify how log files generated by the production system of an SMS broker can be utilized to identify anomalies in RTTs between itself and several external operators.

5.5.1 Data collection

We examined data from the Enterprise Messaging Gateway (EMG), an Infoflex Connect AB product used by many SMS brokers. The data was taken from existing log files as they contained the data we needed without requiring modifications to the core product with a risk of introducing bugs. The amount of data per operator varied between 33 and 497 MB.

In this study we selected one of the most commonly used protocols for SMS traffic, SMPP (Short Message Peer to Peer). Each PDU starts with a header, comprised of the operation number, a transaction number, a status and

ALGORITHM 1: Find Outlier Clusters

```
input : A list of data points, each one consisting of a list of key-value pairs
        and a measured value $V_n$.
output: A list of start and end points for anomalies.
forall A do A $\longleftarrow \emptyset$          // Clear all Remedian arrays.
haveReported $\longleftarrow$ false; (outliers, expected) $\longleftarrow \emptyset$
foreach possible key do
    foreach value used by key do adjustments[key,value] $\longleftarrow 0$
end
foreach data point dp do
    // Update the expected value from
       measurement(dp), minus adjustment factors.
    $b \longleftarrow 0$
    foreach (key,value) in dp do $b \longleftarrow b +$ adjustments[key,value]
    expected $\longleftarrow \alpha *$ (measurement(dp) $- b) + (1 - \alpha) *$ expected
    // Update the adjustment factors.
    foreach (key,value) in dp do
        // Assume all other adjustment factors are
           correct, and calculate what is left.
        diff $\longleftarrow$
         measurement(dp) $-$(expected$+b-$adjustments[key,value])
        // Update the adjustment factor for this
           key-value pair.
        adjustments[key,value] $\longleftarrow$
         $\alpha *$ diff $+ (1 - \alpha) *$ adjustments[key,value]
    end
    i $\longleftarrow 0$                  // Update the Remedian arrays.
    Append expected to A[0]
    while A[i] is full and i $+ 1 < 6$           // We have 6 arrays
     do
        Append median(A[i]) to A[i +1]
        A[i] $\longleftarrow \emptyset$; i $\longleftarrow$ i $+ 1$
    end
    // Find start and end points for anomalies.
    if not haveReported and A[5] has been filled at least once and
     median(A[3]) $> 2 *$ median(A[5]) then
        Add ('start', timestamp(dp)) to outliers
        haveReported $\longleftarrow$ true
    end
    if haveReported and median(A[3]) $<$ median(A[5]) then
        Add ('end', timestamp(dp)) to outliers
        haveReported $\longleftarrow$ false
    end
end
return outliers
```

the length of the data section. Following the header is the data section, consisting of a sequence of key-value pairs, where the keys and their order depend on the operation. As responses can arrive in an undefined order, the transaction number from the request must be exactly duplicated in the response.

The EMG log files contain information on whether each PDU was sent or received, the timestamp, which connection was used, the operation name, the transaction number, and all key-value fields from the data section.

5.6 Case Study Results

This section presents the results of the industrial case study. In particular, we first discuss how different characteristics (keys) of the data and messages sent affect the RTT (Section 5.6.1). Second, we discuss how using certain adjustment factors enabled higher accuracy in the outlier detection (Section 5.6.2). Third, we detail the results of applying the anomaly detection algorithm to a large dataset of network traffic (Section 5.6.3).

In the presented results, data is analyzed for three different operators, referred to as “O1”, “O2” and “O3”.

5.6.1 RTT for selected keys

To explore how individual keys affected the RTTs, we counted the number of unique values used by each key. This revealed three distinct categories.

Message specific: 11 keys, e.g. destination numbers and message bodies. We assume these values are unique for each message.

Groups: 11 keys, e.g. whether a delivery report is requested, the character encoding, and similar keys with a very limited set of values. We identified “data coding”, “esm class” and “registered delivery” as having the largest effect on the RTTs.

Constants: 4 keys that are either not supported by EMG, or ignored by most recipients, and therefore always sent with the same value.

Next we describe the key “data coding” in more detail, and how its value affects the RTT. All RTT values in this section are shown with their mean and one standard deviation up and down, to give an indication of their relative positions and spread.

Table 5.1 shows the RTT grouped by data coding. The values in the first column have the following meaning.

Value	**O1**	**O2**	**O3**
0	9.3/9.7/10	8.1/8.2/8.4	440/466/493
8	23/25/28	21/24/27	651/673/697
240	3.6/3.7/3.7	3.5/3.5/3.5	N/A
245	3.2/4.9/7.7	N/A	329/358/390

Table 5.1: RTT in milliseconds, grouped by data coding. The three values are $\mu - \sigma$, μ and $\mu + \sigma$, respectively.

0 Text message, using the GSM character encoding IA5.

8 Text message, using the character encoding UCS-2.

240 Special messages, e.g. configuration settings.

245 8 bit data, e.g. ring tones.

With the exception of the values 240 and 245 to operator “O1”, the RTT distributions for different values are clearly separated. The operators seem to perform some time consuming processing of UCS-2 texts, as those RTTs are significantly longer than for IA5 texts. “N/A” means the value was not used with that operator.

The RTTs when grouped by the “esm class” and “registered delivery” keys showed similar patterns, with a ratio of up to 3 for some values. This motivates us to show the results with a deeper analysis using the adjustment factors.

5.6.2 Adjustment factors

The adjustment factors for the message key values were mostly consistent with the results in Section 5.6.1. The “data coding” adjustment factors are shown in Table 5.2. As the values represent the difference in exponent, a value of 1 corresponds to a ratio between the RTTs equal to e.

For O1, whether data coding is 0 or 8 gives an RTT that varies by a factor of $e^{0.92-(-0.46)} \approx 3.97$. UCS-2 data requires twice as much space as IA5, but even if we adjust for this, there is still a remaining factor of $3.97/2 \approx 1.99$. We see a similar pattern for O2, with adjustment factors -0.13 and 1.30 for data codings 8 and 240. The “esm class” and “registered delivery” keys also showed a clear correlation between the RTTs and the adjustment factors.

Value	**O1**	**O2**	**O3**
0	**-0.46** (9.7)	-0.13 (8.2)	-0.10 (466)
8	**0.92** (25)	1.30 (24)	0.11 (673)
240	-1.12 (3.7)	-0.87 (3.5)	N/A
245	-0.03 (4.9)	N/A	-0.25 (358)

Table 5.2: Adjustment factors, by data coding. The value inside parentheses is μ from Table 5.1.

5.6.3 Anomaly frequencies

Figure 5.3 uses blue circles to show the RTTs for 288,515 outgoing requests to O1, over a period of approximately two months. Most measurements are around 10 milliseconds (1e+04 microseconds), but RTTs of up to several seconds are common enough that they are not considered outliers. The black, green and red lines show the medians from A_1, A_3 and A_5, respectively, as described in Section 5.4.4. The black line shows the median of the $5^2 = 25$ most recent measurements. Even with the large number of measurements above 1e+06 microseconds at Index 240,000, there is still enough data with lower values to keep the median below 1e+05 microseconds. The green line shows the median of 25 values from the black line, i.e. $25^2 = 625$ measurements. It stays significantly calmer, peaking only for indices 18,000, 190,000 and around 240,000, all corresponding to wider peaks of the black line. The red line shows the median yet another factor of 25 up, for $25^3 = 15,625$ measurements. Although some noise remains, the values shown by the red line (A_5) can be used for comparisons with those shown by the green line (A_3).

The intervals that satisfy our condition for anomalies, i.e. when the median of A_3 is more than twice the median of A_5 as described in Section 5.4.4, are marked with red lines at the bottom of the graph, surrounded by grey dotted rectangles. These lines perfectly mark the sections with many slow responses.

Despite the large variance shown in Section 5.6.1, using adjustment factors and medians provides a base level that is relatively stable. The area containing outliers for O2 is shown in Figure 5.4(b), where the blue dots have been removed for clarity. The end point of the marked area is quite far away from the starting point, indicating low precision of our method. This is the trade-off for high recall and avoiding multiple adjacent groups of outliers. There are no round-trips at 196,000 shorter than 5000 microseconds, causing A_3 (shown by the green line) to increase from 4267 microseconds to 8229. This makes A_3 more than twice the value of A_5 (shown by the red line), i.e. $8229 > 2 * 3964$, satisfying our condition for outliers.

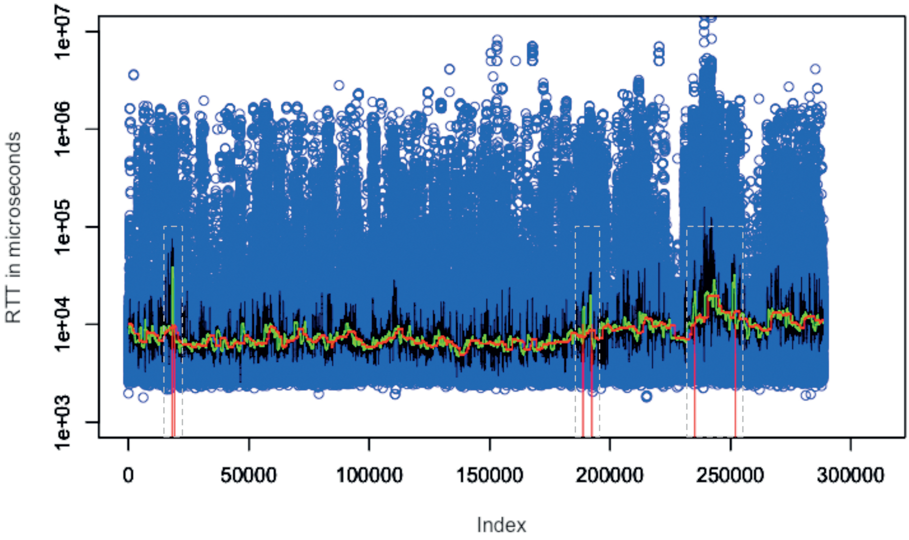


Figure 5.3: RTT and medians for O1.

The effect of the adjustment factors is illustrated in Figure 5.4(a) and Figure 5.4(b). Both figures show the same data, without and with adjustment factors, respectively. The black and green lines in Figure 5.4(b) are more stable, reporting one anomaly instead of three.

The algorithm detected no anomalies in the traffic towards O3.

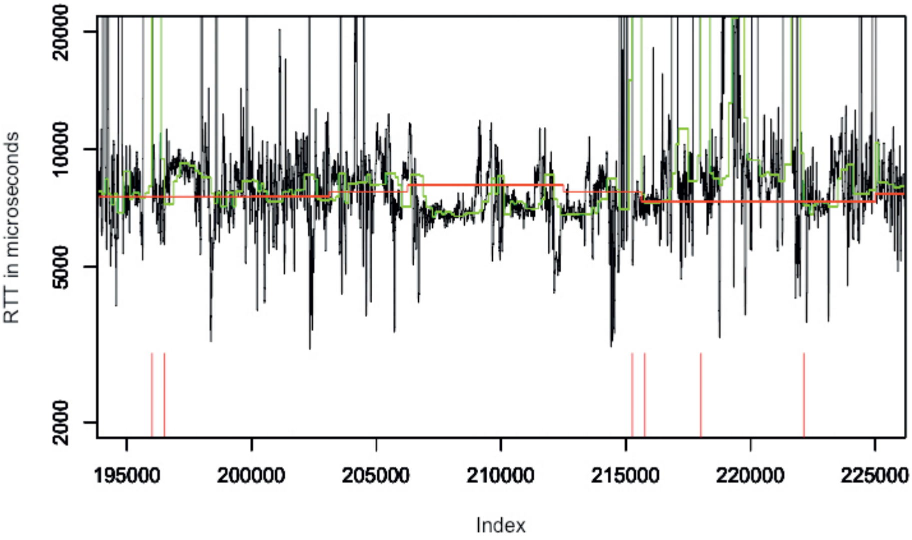
For validation, we created simulated log files. The RTTs were randomized with a log normal distribution and a minimum value of 1000 microseconds. After at least 20,000 roundtrips, a group of up to 4095 entries with up to half a second slower responses was added. The results from the analysis on one such file are shown in Figure 5.5. There were three groups with slow responses, one at 48,515 with 2488 entries, one at 82,120 with 1222 entries, and one at 9133 with 192 entries, corresponding to the three blue peaks. Given there must be at least $625/2 = 313$ entries for our algorithm to report an anomaly, only the first two peaks are reported.

The red line is almost perfectly flat, showing the Remedian [20] is stable.

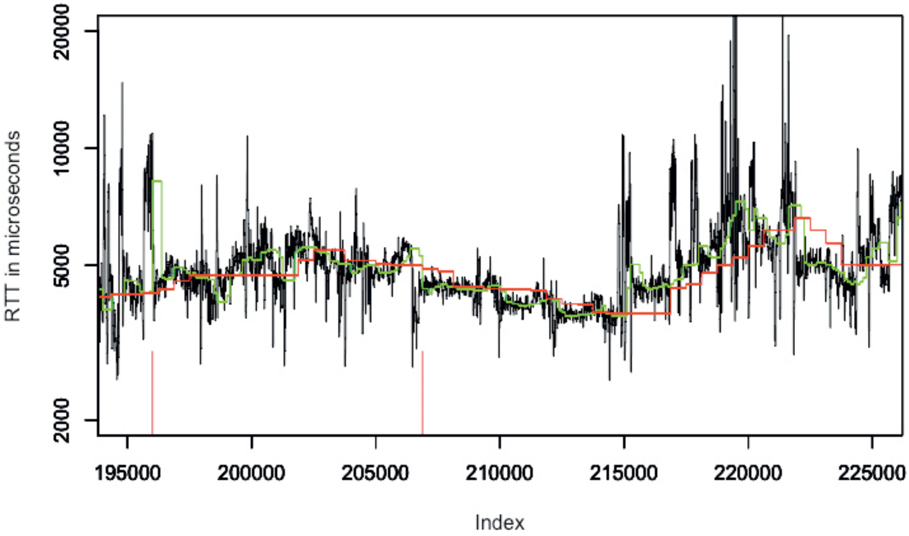
5.7 Validity Threats

Below we discuss the threats to the validity of our study.

Internal: We see two possible internal validity threats. First, although the 8 option keys we discarded in Section 5.6.1 showed no significance in



(a) RTT and medians for O2, without adjustment factors.



(b) RTT and medians for O2, with adjustment factors.

Figure 5.4: RTT and medians for O2 with and without adjustment factors.

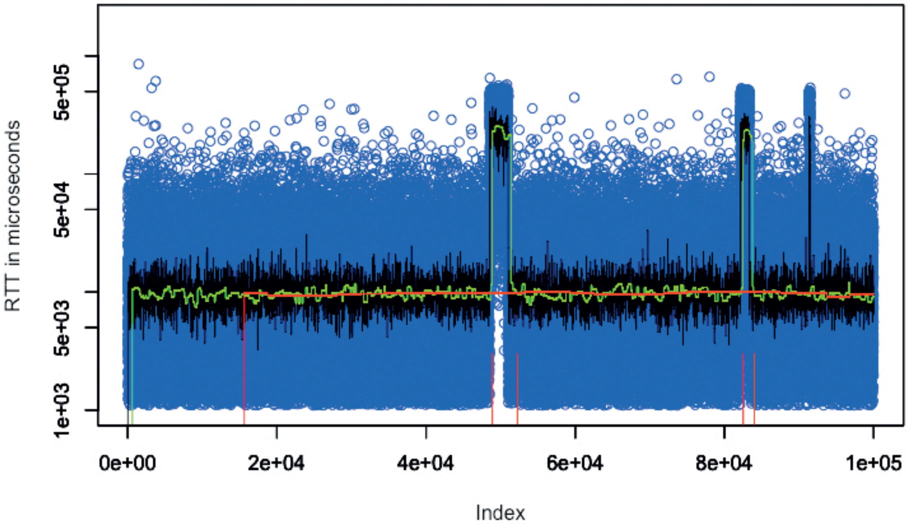


Figure 5.5: Simulated RTTs, with medians and outliers.

the RTTs in our preliminary results, a more advanced analysis might show an effect. Second, any implementation errors were mitigated by carefully examining the program output, manually comparing it with the raw data in the log files.

External: The analyzed log files in this paper all contain SMS traffic over SMPP, but the approach with exponential smoothing and the equations in Section 5.4.2 should be usable in any system where parameter values affect which parts of the code are executed, and therefore also the response time. When calculating the median, the sensitivity can easily be changed by adjusting the array length, and selecting which arrays to compare.

Reliability: We consider the reliability threat to be small, as we have seen similar RTT distributions for connections to several operators around the world.

Construct: The system model used in this paper is somewhat simplified, abstracting the network traffic into logged "send" and "receive" events. In reality, an outgoing PDU requires multiple steps:

1. The data structure with the information to be sent is created.
2. The data is packed into a byte array that can be transmitted on a socket.

3. The data in the data structure is logged.
4. The byte array is sent to the operating system kernel.
5. The operating system sends the byte array to the network device.
6. The network device transmits the byte array to the network.

The timestamp used for the PDU comes from step 3, ignoring any delays caused by the subsequent steps. The operating system used is Unix, which does not provide a simple way to find the exact time for step 6. Instead we assume that delays are small compared to the network transmission and application processing times.

A limitation of the Remedian (described in Section 5.4.4) is that only value sequences that start on multiples of 5^n are considered, so the number of outliers required to trigger an anomaly notification varies. We do not consider this a problem, as the algorithm must always be adapted in order to achieve the desired sensitivity.

5.8 Conclusions and Future Work

Anomaly detection in production systems is valuable for ensuring service levels towards customers. Making use of our domain knowledge, we developed an algorithm that reduces noise, enabling the detection of larger clusters of outliers. The algorithm is implemented as an extension to our tool ELFA which calculates RTTs between different communicating systems.

Even when the average RTT is within acceptable limits when analyzing data from the live production environment of an SMS broker, our approach can be used to identify conditions which have an unreasonable effect. A relevant example would be the UCS-2 handling (see Table 5.2, the factor when the parameter is 8) by O1 and O2. Whilst being functionally correct, it suggests the UCS-2 handling could possibly be made more efficient in those systems.

RTT anomalies may also be possible to detect by observing their side effects, such as a queue of outgoing messages being formed. The higher the throughput normally is, the longer the queue can be while maintaining an acceptable delivery time, so such an algorithm would have to take the current throughput into account. The throughput is not logged by EMG, so we could not use this method.

Acknowledgments

This work was sponsored by The Knowledge Foundation industrial PhD school ITS ESS-H, 20160139 (TestMine), 20130085 (TOCSYC) and Infoflex Connect AB.

Bibliography

[1] S. Agarwala, F. Alegre, K. Schwan, and J. Mehalingham. E2EProf: Automated end-to-end performance management for enterprise systems. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2007.

[2] J. Aikat, J. Kaur, F. D. Smith, and K. Jeffay. Variability in TCP round-trip times. In *Proceedings of the SIGCOMM Conference on Internet Measurement (IMC)*. ACM, 2003.

[3] J. H. Bickford. *Mechanisms for intermittent motion*. Krieger Pub Co, 1972.

[4] D. Brahneborg, W. Afzal, and A. Čaušević. A Black-Box Approach to Latency and Throughput Analysis. In *Proceedings of the Conference on Software Quality, Reliability and Security Companion (QRS)*. IEEE, 2017.

[5] D. Brauckhoff, K. Salamatian, and M. May. Applying PCA for Traffic Anomaly Detection: Problems and Solutions. In *Proceedings of the Conference on Computer Communications (INFOCOM)*. IEEE, 2009.

[6] J. D. Brutlag. Aberrant behavior detection in time series for network monitoring. In *LISA*, volume 14, pages 139–146, 2000.

[7] T. Finch. Incremental Calculation of Weighted Mean and Variance. 2009.

[8] M. Gutiérrez, W. Steiner, R. Dobrin, and S. Punnekkat. Learning the Parameters of Periodic Traffic Based on Network Measurements. In *Proceeings of the International Workshop on Measurements & Networking (M&N)*. IEEE, 2015.

[9] E. Halepovic, J. Pang, and O. Spatscheck. Can you GET me now? Estimating the time-to-first-byte of HTTP transactions with passive measurements. In *Proceedings of the SIGCOMM Conference on Internet Measurement (IMC)*. ACM, 2012.

[10] A. Holst and B. Bjurling. A Bayesian Parametric Statistical Anomaly Detection Method for Finding Trends and Patterns in Criminal Behavior. In *Proceedings of the European Intelligence and Security Informatics Conference*, 2013.

[11] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose, and D. Towsley. Inferring TCP Connection Characteristics Through Passive Measurements. In *IEEE INFOCOM*, volume 3, pages 1582–1592. IEEE, 2004.

[12] H. Jiang and C. Dovrolis. Passive estimation of tcp round-trip times. *ACM SIGCOMM Computer Communication Review*, 32(3):75–88, 2002.

[13] A. Lakhina, M. Crovella, and C. Diot. Mining anomalies using traffic feature distributions. *ACM SIGCOMM Comp. Comm. Rev.*, 35(4), 2005.

[14] Y. Liu, L. Zhang, and Y. Guan. Sketch-based Streaming PCA Algorithm for Network-wide Traffic Anomaly Detection. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2010.

[15] H. Martin, A. McGregor, and J. Cleary. Analysis of Internet Delay Times. Technical report, 2000.

[16] A. F. M. Mellia and M. M. M. M. Munaf. Experiences of Internet Traffic Monitoring with Tstat. *IEEE Network*, (June):8–14, 2011.

[17] D. Mosberger and T. Jin. httperf - A Tool for Measuring Web Server Performance. *SIGMETRICS Performance Evaluation Review*, 26(3):31–37, 1998.

[18] V. Paxson. Automated Packet Trace Analysis of TCP Implementations. *ACM SIGCOMM Computer Communication Review*, 27(4):167–179, 1997.

[19] H. Ringberg, A. Soule, J. Rexford, and C. Diot. Sensitivity of PCA For Traffic Anomaly Detection. *ACM SIGMETRICS Performance Evaluation Review*, 35(1), 2007.

[20] P. J. Rousseeuw and G. W. Bassett. The Remedian: A Robust Averaging Method for Large Data Sets. *Journal of the American Statistical Association*, 85(409):97–104, 1990.

[21] S. Shanbhag and T. Wolf. Accurate Anomaly Detection Through Parallelism. *IEEE Network*, 23(1):22–28, 2009.

[22] M. Steinder and A. S. Sethi. A Survey of Fault Localization Techniques in Computer Networks. *Science of Computer Programming*, 53(2):165–194, 2004.

[23] J. W. Taylor. Short-Term Electricity Demand Forecasting Using Double Seasonal Exponential Smoothing. *Journal of Operational Research Society*, 54(8):799–805, 2003.

[24] J. W. Tukey. *Exploratory Data Analysis*. Reading, MA, 1977.

[25] C. Wang, K. Viswanathan, L. Choudur, V. Talwar, W. Satterfield, and K. Schwan. Statistical Techniques for Online Anomaly Detection in Data Centers. In *Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2011.

[26] H. Yan, K. Li, S. Watterson, and D. Lowenthal. Improving Passive Estimation of TCP Round-trip Times Using TCP Timestamps. In *Proceedings of the International Workshop on IP Operations and Management*. IEEE, 2004.

[27] S. Zander and G. Armitage. Minimally-intrusive Frequent Round Trip Time Measurements Using Synthetic Packet-pairs. In *Proceedings of the Conference on Local Computer Networks (LCN)*, 2013.

[28] H. Zeng, M. Di Natale, P. Giusto, and A. Sangiovanni-Vincentelli. Using Statistical Methods to Compute the Probability Distribution of Message Response Time in Controller Area Network. *IEEE Transactions on Industrial Informatics*, 6(4):678–691, 2010.

Paper B.
A Lightweight Architecture Analysis of a Monolithic Messaging Gateway

A Lightweight Architecture Analysis of a Monolithic Messaging Gateway. Daniel Brahneborg, Wasif Afzal.

In *Proceedings of the International Conference on Software Architecture (ICSA)*. IEEE, 2020.

Abstract

Background: The Enterprise Messaging Gateway (EMG) from Infoflex Connect (ICAB) is a monolithic system used to deliver mobile text messages (SMS) world-wide. The companies using it have diverse requirements on both functionality and quality attributes and would thus benefit from more versatile customizations, e.g. regarding authorization and data replication.

Objective: ICAB needed help in assessing the current architecture of EMG in order to find candidates for architectural changes as well as fulfilling the needs of variability in meeting the wide range of customer requirements.

Method: We analysed EMG using a lightweight version of ATAM (Architectural Trade-off Analysis Method) to get a better understanding of how different architectural decisions would affect the trade-offs between the quality requirements from the identified stakeholders.

Result: Using the results of this structured approach, it was easy for ICAB to identify the functionality that needed to be improved. It also became clear that the selected component should be converted into a set of microservices, each one optimized for a specific set of customers.

Limitation: The stakeholder requirements were gathered intermittently during a long period of continuous engagement, but there is a chance some of their requirements were still not communicated to us.

Conclusion: Even though this ATAM study was performed internally at ICAB without direct involvement from any external stakeholders, documenting elicited quality attribute requirements and relating them to the EMG architecture provided new, unexpected, and valuable understandings of the system with a rather small effort.

6.1 Introduction

Mobile text messages (SMS) are used world-wide, being popular as they work on all mobile phones without any additional software installed. In particular, they are commonly used by companies to send meeting reminders, authentication codes, travelling tickets, and more, to their customers. Between these companies and the customers' network operators, we find a product segment called SMS gateways. These gateways receive text messages from the companies, route them to the right operators, manage the connections to the operators over several different communication protocols, and handle any operator specific requirements. The gateways are often run by a separate group of companies, known as SMS brokers.

One of these gateways is the Enterprise Messaging Gateway (EMG) from Infoflex Connect AB (ICAB). EMG is a system with a proven track record spanning more than 20 years. Its monolithic architecture makes it easy for customers to deploy and manage EMG, and is also convenient for the software developers as the entire code base is just a simple function call away.

However, monoliths can be difficult to scale horizontally, i.e. to more than one server, and are sensitive to failures as those can bring the entire application down [22]. In our context, it is also problematic that each software update requires a full application restart, temporarily stopping all traffic.

One common way of addressing some of these issues with monoliths is to split them into sets of microservices [6, 14]. Selecting which parts of the monolith to extract is non-trivial [12, 10], but usually involves identifying components with loose coupling (independent) and strong cohesion (self-contained) [17].

Older literature on modularity provide some other recommendations, such as "*We propose instead that one begins with a list of difficult design decisions or design decisions which are likely to change. Each module is then designed to hide such a decision from the others.*" by Parnas [18]. We find a similar theme in many of the design patterns by Gamma *et al.* [11], namely to "*encapsulate the concept that varies*".

The approach used by Cruz *et al.* [7] was to base their migration of a monolith to microservices on the results of an architectural analysis of the system. Such an analysis focuses on aspects which arise as a consequence of the architecture, e.g. response times, scalability, and modifiability. In particular, they used the Architectural Trade-off Analysis Method (ATAM) created at the Carnegie Mellon's Software Engineering Institute [16, 3]. According to both Dobrica *et al.* [8] and Anjos and Zenha-Rela [2], the only analysis methods that consider a wide set of quality attributes are ATAM and SBAR [4]. Of these, ATAM is the only one applicable to mature products [2]. A more recent approach is RCDA, the Risk- and Cost Driven Architecture approach [19], adding a financial dimension to architectural work. RCDA appears to be most useful early in relatively large projects, neither of which is the case for us.

We know from experience that software such as EMG, created by one company and used by others, usually require variability in terms of customer specific behaviour. This variability is typically not required for microservices, as they tend to be created for in-house use and even operated by the developers themselves, sometimes phrased as "*you build it, you run it*" [13]. This in-house focus means that "*the literature [on microservices] is scarce in relation to the use of variability*" [6]. Variability is also not covered in the otherwise comprehensive mapping study on microservice architecture by Alshuqayran *et al.* [1].

The research questions we address in this work are whether ATAM could help identifying the components in EMG where architectural changes would be most beneficial, and whether it can help clarifying the need for variability in those components. During the analysis, all known quality requirements as given both by ICAB and their customers must be taken into account.

We present the results of the systematic application of a lightweight version of ATAM to ICAB's EMG system. Our primary contribution is showing that the ATAM analysis was able to identify the best component to change, which in ICAB's case is the credit management. Our second contribution is showing that ATAM also helped with managing the variability, suggesting that the credit management component should be extracted into a set of microservices, all providing the same API but implementing different strategies. Our third contribution is the overview of the ATAM artefacts and concepts (Fig. 6.6), with more details than the original paper [16]. Our fourth contribution is a description of how ATAM can be used in a real-world architecture analysis, even in the absence of external stakeholders (Section 6.3.5).

The analysis was carried out as an entirely internal project at ICAB, avoiding the overhead that comes with involving external parties. While this setup presented a risk of missing some of the requirements and ignoring the relative importance of the requirements we found, it also meant that the study could be completed in just over one man month.

The current section has presented the background for the analysis and related work. Next, Section 6.2 describes ATAM and how we adapted it for our purposes, Section 6.3 presents the results from the ATAM analysis, and Section 6.4 discusses the interpretation of those results. Section 6.5 discusses the validity threats and finally, Section 6.6 presents our conclusions and planned future work.

6.2 Method

ATAM [16] is a systematic way to “assess the consequences of architectural decisions in light of quality attribute requirements”. We therefore only consider requirements whose measurable responses to external stimuli are affected by the architecture.

An ATAM study follows the nine steps below, typically in a two or three day discussion workshop. The participants of this workshop are the various stakeholders of the system. The analysis is usually carried out in two phases, where phase 1 is limited to steps 1–6 and a small team, and phase 2 includes all steps a few weeks later, now with the full team.

Presentation:

1. Present ATAM to the participants.

2. Present the system from a business perspective.

3. Present the suggested architecture.

Investigation and analysis:

4. Identify architectural approaches. These approaches refer to aspects such as whether the system would be a monolith, client–server, or something else. Other terms used here are “styles” and “patterns”.

5. Generate the quality attribute utility tree, a hierarchical list of quality requirements.

6. Analyze the architectural approaches in step 4, on how they realize the most important quality attributes described in step 5. This results in the identification of sensitivity points where an architectural approach affects a quality attribute, tradeoff points where an approach affects multiple quality attributes in different ways, and the list of such points that present a risk.

Testing:

7. Elicit and prioritize scenarios. Now the full set of stakeholders are included, brainstorming both current use cases, expected future “change scenarios”, and extreme “exploratory scenarios”.

8. Analyze the architectural approaches again, now focusing on the most important scenarios from step 7. If a scenario can not be realized using the selected architectural approaches, these need to be adjusted.

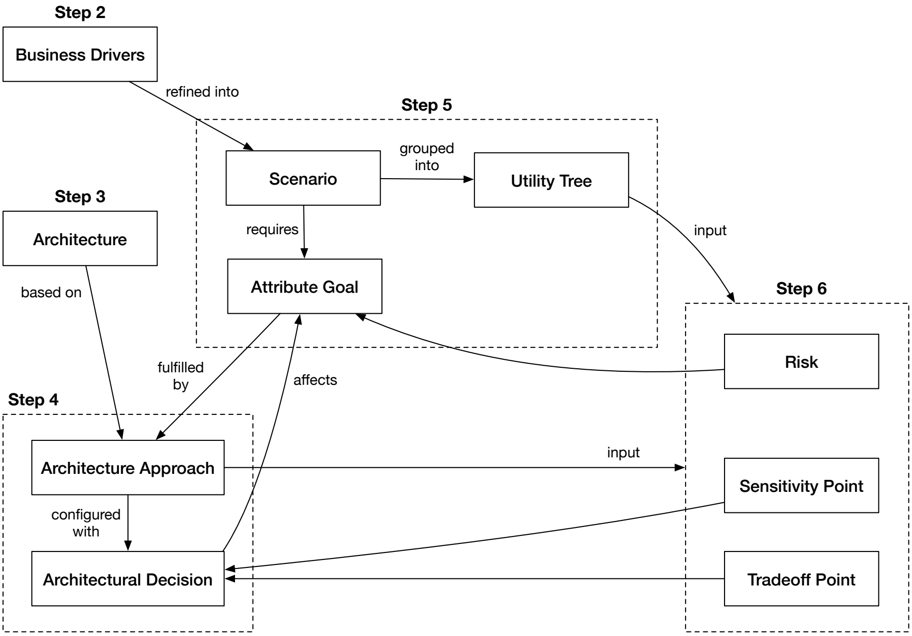


Figure 6.6: Artefacts created by the different steps in the ATAM version used in this paper, with arrows indicating data flow. The analysis output is discussed in detail in Section 6.3.5.

Reporting:

9. Present the results.

The key concepts used in ATAM, according to our understanding, are shown in Fig. 6.6. We noted that there is a discrepancy in the ATAM paper regarding the output from step 6. First it says "*all sensitivity points and tradeoff points should be categorized as either a risk or a non-risk*", suggesting that risk is an orthogonal concept to the first two. However, then it continues "*The risks/non-risks, sensitivity points, and tradeoff points are gathered together in three separate lists*", where the risks are now a concept of its own. In this paper we used the second variant, with the risks list containing attribute goals we do not yet know how to fulfill.

We made some additional adaptations of ATAM to fit our situation better. First of all, no external stakeholders were involved. Therefore, our analysis was based on the lightweight version of ATAM [3], where the testing done in steps 7 and 8 is skipped. This is, in essence, ATAM phase 1 plus reporting. The current section represents step 1 and Section 6.3 represents step 9, containing the output produced by steps 2 to 6.

Next, we realized that the prioritization of the quality attributes in step 6, “Analyze Architectural Approaches”, was highly dependent on which stakeholders were present. To get an objective result despite the lack of such stakeholders, we needed another way of identifying the most important scenarios. For this purpose, we counted the number of business driver groups interested in each of the quality attributes. The point of having a varied set of stakeholders would partly be to have somebody represent all these different groups, making this simple count seem a reasonable proxy.

6.3 Results

This section represents step 9, presenting the output generated from steps 2 to 6. We recall that step 1 was covered by Section 6.2, and that steps 7 and 8 were skipped entirely.

6.3.1 Step 2: Business Drivers

The system requirements from a business perspective were grouped into “important functions”, “major quality attribute goals”, “business goals”, and “constraints”.

- The most **important functions** of EMG are to:
  - forward messages with high throughput, and
  - manage client credits for these messages.
- The **major quality attribute goals** of EMG are to:
  - be available without interruptions,
  - prevent data loss, and
  - be easy to install and maintain.
- To satisfy the **business goals**, each EMG release should:
  - provide increased value to a varied set of both old and new customers, and
  - be done several times per year, to minimize risks for both ICAB and the SMS brokers by not containing too many new features or updated behaviours.
- The **constraints**, defining the border between what can and what can not be done, say that:

- EMG must follow standard network protocols (e.g. SMPP and HTTP),
- the relative message ordering does not need to be maintained [5], and that
- ICAB does not have the resources for a full software rewrite. This means focusing on the smallest changes in the most isolated modules, giving the most value to the largest number of customers.

Step 2 also includes the identification of the major stakeholders, which in our case consists of three groups. The first group consists of the staff at ICAB, where there is a strong focus on maintaining a sound architecture, making the required effort for adding new features predictable. In the second group we have the entry level customers, operating gateways with a limited amount of traffic. These customers run EMG on a single node, trading potentially lower availability for a lower cost and a simpler setup. The third and final group consists of customers using multiple EMG nodes and complex configurations to achieve higher system-level throughput, higher availability to clients, and better protection of queued messages.

6.3.2 Step 3: Analyzed Architecture

At the highest abstraction level, EMG, installed at an SMS broker, sits between one or more clients on one side, and one or more mobile network operators on the other side. The clients send messages to EMG, the messages are routed within EMG according to the site specific configuration, and persisted on disk until they can be sent to the designated operator. After a message has been acknowledged by the operator, it is removed from disk. Additionally, the operator can send back message specific delivery reports, which are handled much the same way as normal messages, but travelling in the opposite direction.

To maximize the throughput whilst minimizing the complexity of the installation and configuration, EMG is a modular monolith. Optionally, customer specific plugins can be used at a handful of well defined points in the message life cycle.

EMG normally only needs to be restarted for installation of optional software updates, typically less than a few times per year. In order to prevent data loss and make these restarts as opaque as possible for the clients, it is critically important that the restarts are handled correctly. To accomplish this, the modules in EMG are grouped into three types based on how their data is managed. The modules without persistent state comprise the first type. The second type uses an embedded NoSQL database (currently LevelDB from Google), and the third type uses an external MySQL database.

Modules of the first type provide functionality such as network connectivity and protocol drivers (e.g. SMPP and HTTP), address and content filtering and modification, message routing, and logging. Most of these modules can use either configurable builtin logic or a site specific plugin, developed either by ICAB or the customers themselves.

The data managed by the second type of modules, using an embedded database, is not accessible from the outside. This enables ICAB to change both the actual database used as well as the structure of the stored data, as needed. These modules primarily handle the message queues and information about pending delivery reports. There is also a module for the SAT (Source Address Translation) functionality, providing a dynamic mapping between the incoming sender address from the client and the address used towards the operator. This functionality enables a client to use an email address for the sender address, which is mapped to a phone number picked from a number pool. The message recipient can then reply to this pool number, which the SAT module converts back to the original email address.

The third type of modules manage data using the MySQL client API, making the data available for modification from the outside. This gives SMS brokers the option to use either a simple single node database or a multi-node replicating cluster, without requiring any special handling by EMG. These modules mainly handle user authentication and message credits. There is usually also a read-only view of the last known state (e.g. received, forwarded, failed, or delivered) of each accepted message, used for billing and troubleshooting.

6.3.3 Step 4: Architectural Approaches

At the top level, EMG is best described as using the publish-subscribe [9] architectural style. Even though “connector” is part of the publish-subscribe style, in the EMG context it is used as an endpoint definition from clients or to operators. The embedded NoSQL storage acts as the event bus, the connection between producers and consumers. The publisher is driven by the incoming connector the client connects to, and the consumer is driven by the outgoing connector. The events are the text messages, and the event types correspond to the names of the connectors. Each event must only be received by a single consumer, to avoid multiple copies of each message appearing in the recipients’ mobile phones, barring exceptional circumstances.

Both producers and consumers use a simple layered approach. The data

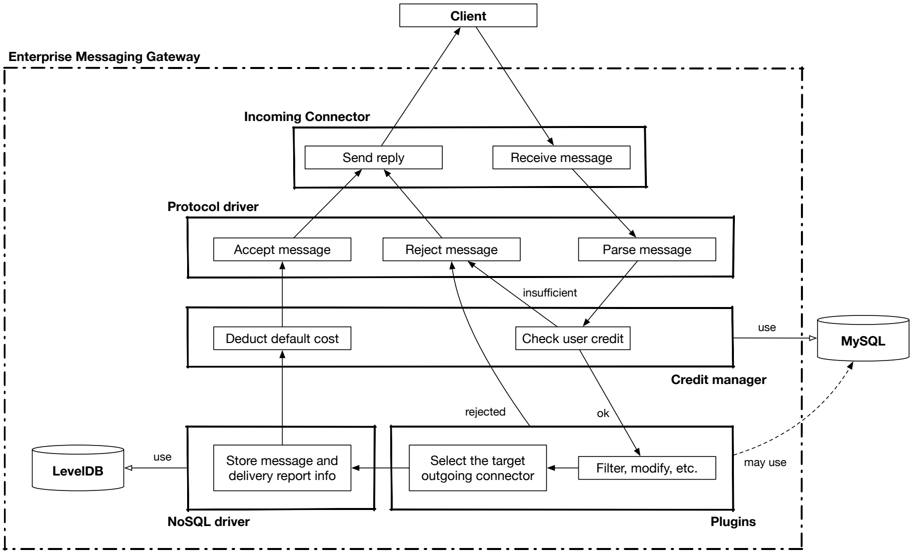


Figure 6.7: The main data flow for incoming traffic from clients.

flow of the producer side is shown in Fig. 6.7, where each received message passes down through various modules before a response propagates back to the sender. The incoming connectors receive messages using one of the protocol drivers, filter, modify and route them as configured, and finally persist them in the NoSQL storage. Similarly, the outgoing side handles connectivity to the systems downstream, forwards the messages, removes them from storage, and finally adjusts the user credit value. The usage of an embedded database provides good performance and protection from temporary node failures, but not from permanent node failures.

EMG is deployed on one or more, physical or virtual, 64 bit Linux nodes. The deployment requires the installation of a number of third party components, adding the executable files for EMG, and making site specific configurations. MySQL is deployed on the same or other nodes.

Using multiple EMG nodes is beneficial despite the lack of data replication between them, as permanent node failures are rare. Separate sets of messages can then be processed in parallel by all nodes, resulting in increased system-wide throughput.

Based on the assumption that the data in the embedded databases is limited in size, all such data is loaded on startup into specifically tailored in-memory data structures. Primarily, this frees the “find the next message to send” and “find the delivery report record corresponding to a given message id” operations from having to make time consuming round-trips to the database. These operations can now be carried out much faster by instead traversing the data structures.

6.3.4 Step 5: Quality Attribute Tree

In step 5, the quality attributes are elicited, prioritized, and refined with exact stimuli and falsifiable responses into scenarios, which are to be analyzed in the next step. Typically these attributes originate from the business goals, the result from step 2. ATAM provides a sample list of possible quality attributes, but also notes that stakeholders "*may add their own quality attributes or may use different names for the same ideas*" [16]. Another attribute list for use in ATAM is presented by Bass *et al.* [3]. We decided to base our list on SIS-ISO/IEC 9126 [21], because it had a focus on measurable attributes.

Defining all scenarios fully was not considered meaningful in this paper. For example, the functional requirement that all incoming messages should be forwarded can be refined with the quality requirement that this should be done as soon as possible, which in turn could be refined by setting a maximum time limit. However, the time a message is spent queued within EMG depends entirely on the availability of and throughput to the receiving side, both of which are beyond our control.

The full list of elicited quality attributes for the current use cases are summarized together with their corresponding business drivers in Fig. 6.8, and described in more detail below. The attributes selected for step 6 are written in boldface.

6.3.4.1 Functionality

The database containing the current state of each message should be kept reasonably up-to-date. As this requirement was not among the ones analyzed in the next step, the exact limit for the allowed delay was not specified further.

In order for the EMG owner to be able to bill the clients correctly, and possibly also ensure messages are pre-paid, the current value of the message credits for each user must be kept up-to-date. A discrepancy of 1 second's worth of messages would be acceptable, if that increases the throughput.

Ever since EMG was created, it has followed the general robustness principle of being as tolerant as possible when processing incoming data, and as conservative as possible when producing outgoing data. Following this principle ensures a minimum of code changes are required for EMG to exchange messages with other systems, and has served the product well.

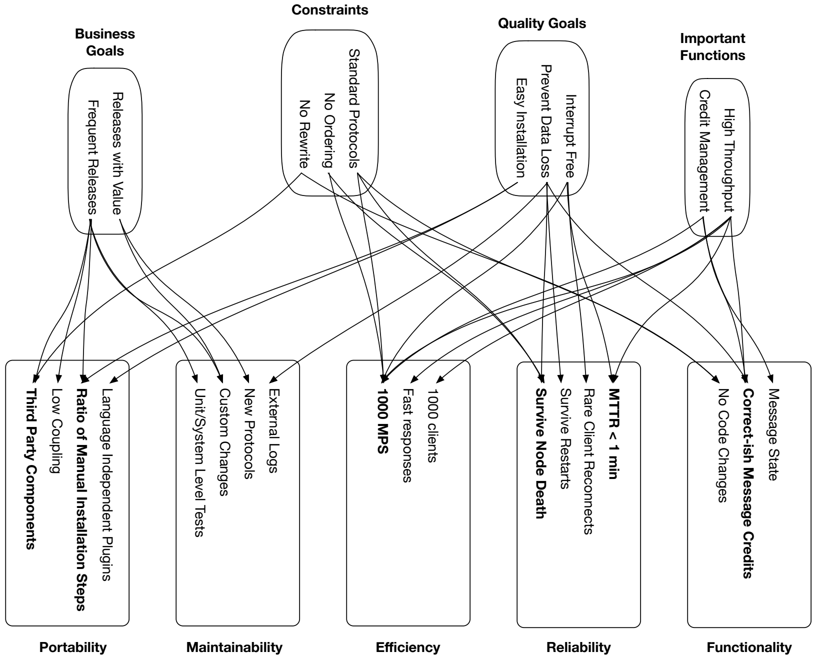


Figure 6.8: Business Drivers (top) and Quality Attributes (bottom) for EMG.

6.3.4.2 Reliability

There are two types of quality attribute requirements in the reliability category. The first type concerns the availability to clients, stating that forcing clients to reconnect to EMG should be a rare event, and happen at most once per several months. Additionally, when such an interruption occurs, for example due to EMG being restarted, connectivity should be restored in less than one minute.

The second type concerns the prevention of message loss. As there is no dependable end-to-end acknowledgement for text messages, a message received and confirmed back to the sender must remain in the queue and eventually get sent, regardless of whether the EMG application stops temporarily or permanently. The latter is of course only possible to achieve in a configuration with multiple nodes and data replication.

6.3.4.3 Efficiency

Most modules in EMG are I/O bound, either for disk, network, or both. In order to effectively utilize modern hardware with fast multi-core processors, each EMG node must support traffic from at least 1000 parallel client connections. Moreover, a majority of clients send very few messages, which means a large number of clients are required for the SMS gateway node to be profitable.

Next, incoming messages should result in an acknowledgement sent back within 10 seconds. Some SMS gateways wait up to a minute before sending back a reply, making it difficult for the sender to know whether the remote system is still operational. An EMG running on a local node has no problems replying within 1 millisecond, but this time increases significantly if routing is done using a plugin requiring one or more network roundtrips to other systems, or if the message must be replicated to a different data center.

Finally, each EMG node should be able to process at least 1000 messages per second (MPS). This number is based on both the requirements from existing EMG customers, and what seems to be reasonable given the current architecture. The highest throughput is typically reached for 10–100 parallel connections, as this keeps all processor cores busy.

6.3.4.4 Maintainability

For maintainability, where the requirements are based almost entirely on the business goals of doing frequent and valuable releases, we consider both development and operational perspectives. For the development perspective, adding new protocols and custom logic should not affect the rest of the system. This is supported by a comprehensive regression test suite. For the operational perspective, the most important requirement is to assist troubleshooting by having EMG log over the network.

6.3.4.5 Portability

The portability requirements concern the independence of the system from its surrounding environment, and of its parts relative to each other. To achieve the former, plugins should be possible to implement in any programming language available on the Linux platform, and the number of manual steps in the installation procedure should be kept to a minimum. Until recently, plugins could only be written in C and Perl, but with the recent addition of an HTTP interface they are now language independent. For the installation, having to install multiple system packages can cause conflicts with other applications. Achieving the latter, where the parts are independent from each other, enables making isolated and predictable changes. Keeping system parts independent is easier when using third party components, as they are guaranteed not to have any dependencies back into EMG.

6.3.5 Step 6: Analyze Architectural Approaches

In step 6, the architectural approaches from step 4 are analyzed on how well they support the most important requirements from step 5. As mentioned, we found those requirements by counting the number of business goals they were related to. This way we found "1000 MPS" linked to 3 groups, followed by "correct-ish message credits", "MTTR $<$ 1 min", and "survive node death", each one linked to 2 groups. We also found links to 2 groups from "ratio of manual installation steps" and "third party components". We recall that these requirements are all marked with boldface in Fig. 6.8. Conversely, each one of the business drivers "high throughput", "prevent data loss" and "standard protocols" affects the largest number of quality attribute groups, i.e. 3. We describe the most significant architecture parameters related to these attributes below.

6.3.5.1 Risks

As motivated in Section 6.2, our list of risks contains "*architecturally important decisions that have not been made*" [16]. In our case, the only identified risk concerns the temporary storage of unsent messages. How to ensure messages survive the permanent failure of the node they arrived to, while still maintaining the desired throughput and following the constraints from step 2, is an open issue.

6.3.5.2 Sensitivity points

The sensitivity points are "*parameters in the architecture correlated to measurable quality attributes*" [16]. Production environment log files from various sites clearly show that the greatest effect on the time required to restart the EMG application after a failure, i.e. the MTTR, is due to the preloading of NoSQL data. The lion's share of the startup time is used loading the runtime data for the SAT functionality. This is because the SAT data is kept for several days, whereas messages waiting to be sent and data on pending delivery reports are typically removed after just a few seconds.

The ratio of manual installation steps and the coexistence with third party components would both benefit from improved installation procedures. In particular, using container technology such as Docker[1] could have considerable positive effects.

[1] `https://www.docker.com`

6.3.5.3 Trade-off points

Trade-off points are similar to sensitivity points, but "*correlated to multiple quality attributes with different effects*" [16]. The architectural decision of storing the message credits in an external database is the source of two such points, both related to performance.

The first point concerns the trade-off between performance and usability. By keeping the credits in MySQL, we improve usability as the credits are easy to read and update from an external tool. However, because each database operation takes non-zero time, at the same time we also lose throughput.

The second point is the trade-off between performance and precision. To mitigate the lowered throughput from the previous trade-off, credit updates could be limited to once per n messages. This would improve performance on account of fewer database operations, but reduce the precision of the current credit value. The credit value would be only eventually consistent at each point in time, so there are windows of time in which clients might be able to send more messages than they should be, according to their pre-paid message credits.

6.4 Discussion

Using multiple EMG nodes can not only protect from data loss when an individual node fails, but also increase the total system throughput as more work can be done in parallel. Only the data protection would benefit from replication of the data in the embedded databases, in particular the messages yet unsent. In contrast, both data protection and throughput would benefit from having an effective way of keeping the message credits synchronized among all nodes, as it would allow clients to simultaneously connect to multiple EMG nodes. Furthermore, the ability of doing batch-wise updates of the message credits may provide increased throughput also for the smaller customers using a single EMG node.

We see that improving the credit management would have the biggest impact as it would increase the system throughput, and thereby the value of EMG, for basically all customers. Such an enhancement could be carried out in multiple steps, with each step producing a new variant to be released and then maintained independently. These variants would probably include, but not necessarily be limited to, the ones listed below. Fig. 6.9 shows how they replace the "use" arrow between the credit manager and MySQL to the right in Fig. 6.7.

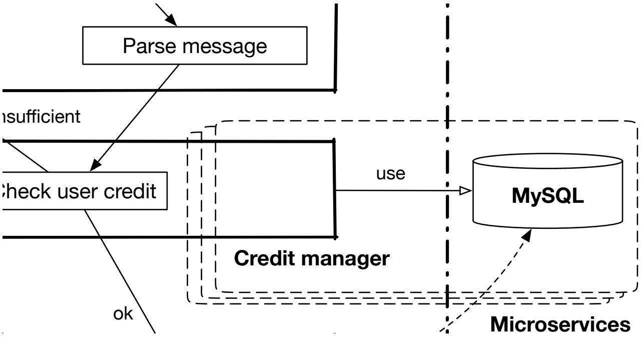


Figure 6.9: The connection between the credit manager and MySQL, shown to the right in Fig. 6.7, would be replaced by one of three microservices.

1. A minimal microservice which checks and updates the credit value for every message, in the same way as the existing implementation.
2. A microservice which updates the credit value with a configurable frequency.
3. A microservice replicating the message credits between EMG nodes, allowing for a more efficient solution with higher throughput than with a replicated database.

The modularity of EMG frees the rest of the system from having to care whether credit updates are batched, if the values are replicated, and if so, how. The components not managing the credit value need only be able to ask if a user can send more messages, and request credit updates when the messages are processed. This loose coupling makes the credit management an excellent candidate for extraction to microservices. As a microservice it can be updated without violating the reliability requirement of allowing clients to remain connected to EMG for extended periods of time. Due to the increased operational complexity of microservices compared to a single monolith, ICAB may also consider implementing the batching of the credit updates as an optional feature in the EMG core.

The answer to our question regarding whether ATAM could help identifying the components in EMG where architectural changes would be most beneficial, is therefore a definite yes. It is also clear that ATAM can help clarifying the need for variability in such components. Additionally, the analysis showed the significance of the SAT implementation. The SAT feature is used by only a few EMG customers, but as the long startup time of the current implementation risks violating the important availability requirement, it needs to be updated.

6.4.1 Lessons learned

The EMG team members are well acquainted with the customer requirements since they all have been involved with the development of the product during its entire lifetime, spanning more than 20 years. The new insights provided by ATAM therefore came a surprise, as the extraction of the credit management into a separate component had never before been considered.

The flexibility of ATAM turned out to be very beneficial to us. As mentioned in Section 6.3.4, the examined quality attributes could easily be adjusted for the specific system being analyzed. Despite ATAM being presented as a tool for improving the communication between stakeholders [16], we were able to adjust it to give meaningful results in ICAB's context even though none of the external stakeholders participated.

We were also able to identify prioritized quality attribute scenarios directly in Fig. 6.8 as described in Section 6.3.5, without having to create an explicit quality attribute tree. This worked in our context as the focus was on highlighting the most prioritized scenarios, rather than making a finer prioritization in terms of "high", "medium" and "low". We appreciate that this allowed us to save both time and effort as utility tree generation is known to be an effort consuming activity [7].

One additional benefit of applying ATAM in our context was the capture of variability scenarios, whereby the component undergoing architectural change would be customized according to the needs of different customers. This explicit identification of variability in architecture perfectly matches ICAB's business goals, and other companies can very well benefit from eliciting such variations without having to go all the way to CBAM (Cost-Benefit Analysis Method) [15, 3].

6.5 Threats to Validity

Runeson and Höst [20] have grouped validity threats into construct, internal, external and reliability. External validity threats concern whether the results are still valid in a more general context. We recognize that some of the simplifications mentioned in Section 6.4.1 fall into this category.

Reliability threats concern whether other researchers would get the same result. The most problematic threat here is the list of business drivers elicited in step 2, and thereby also the list of quality attribute requirements analyzed in step 6, which may both be incomplete. We addressed this by revisiting the lists over several months, adjusting them until they stabilized.

Another reliability threat is the selection of attribute requirements from step 5, where we used a different method than suggested by ATAM. Given what we know about the existing EMG installations, the results still appear reasonable.

6.6 Conclusions and Future Work

The monolithic application EMG needed to provide higher availability and better data protection, which motivated us to performed an ATAM analysis of the existing architecture. EMG is a messaging gateway, which stores and forwards short text messages, and is used by customers with very diverse requirements. As messages are such an important core concept of the application, we had long thought that the replication of those messages would be the next step forward for EMG. Taking all requirements into account, it instead became clear that updating the credit management, used for the billing of the senders of text messages, would provide the most value to the largest set of customers. This improvement will be realized by moving the credit management to a set of microservices, which in turn will allow each customer to select the balance between usability, performance and precision which best suits their particular needs.

For future work, we plan to evaluate the new credit management microservices on relevant quality attributes, e.g. code complexity and performance. ICAB also plans to investigate how best to improve the installation procedures, possibly by making use of container technologies.

Acknowledgments

This work was sponsored by The Knowledge Foundation industrial PhD school ITS ESS-H, Infoflex Connect AB and H2020 project ADEPTNESS (871319).

Bibliography

[1] N. Alshuqayran, N. Ali, and R. Evans. A Systematic Mapping Study in Microservice Architecture. In *Proceedings of the International Conference on Service-Oriented Computing and Applications (SOCA)*. IEEE, 2016.

[2] E. Anjos and M. Zenha-Rela. A Framework for Classifying and Comparing Software Architecture Tools for Quality Evaluation. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 6786 LNCS(PART 5):270–282, 2011.

[3] L. Bass, P. Clements, and R. Kazman. *Software Architecture in Practice, 3rd ed.* Addison-Wesley Professional, 2013.

[4] P. Bengtsson and J. Bosch. Scenario-based Software Architecture Reengineering. In *Proceedings of the International Conference on Software Reuse (Cat. No.98TB100203)*. IEEE, 1998.

[5] D. Brahneborg, W. Afzal, A. Čaušević, and M. Björkman. Towards a More Reliable Store-and-forward Protocol for Mobile Text Messages. In *Proceedings of the Workshop on Advanced Tools, Programming Languages, and PLatforms for Implementing and Evaluating Algorithms for Distributed systems (PODC/ApPLIED)*, New York, NY, USA, 2018. ACM Press.

[6] L. Carvalho, A. Garcia, W. K. G. Assunção, R. Bonifácio, L. P. Tizzei, and T. E. Colanzi. Extraction of Configurable and Reusable Microservices from Legacy Systems: An Exploratory Study. In *Proceedings of the International Systems and Software Product Line Conference (SPLC)*, New York, NY, USA, 2019. ACM Press.

[7] P. Cruz, H. Astudillo, R. Hilliard, and M. Collado. Assessing Migration of a 20-Year-Old System to a Micro-Service Platform Using ATAM. In *Proceedings of the International Conference on Software Architecture Companion (ICSA-C)*. IEEE, 2019.

[8] L. Dobrica and E. Niemelá. A Survey on Software Architecture Analysis Methods. *IEEE Transactions on Software Engineering*, 28(7):638–653, 2002.

[9] G. Fairbanks. *Just enough software architecture: a risk-driven approach.* Marshall & Brainerd, 2010.

[10] J. Fritzsch, J. Bogner, A. Zimmermann, and S. Wagner. From Monolith to Microservices: A Classification of Refactoring Approaches. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 11350 LNCS, pages 128–141. 2019.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional, 1994.

[12] J. P. Gouigoux and D. Tamzalit. From Monolith to Microservices: Lessons Learned on an Industrial Migration to a Web Oriented Architecture. In *Proceedings of the International Conference on Software Architecture Workshops (ICSAW)*. IEEE, 2017.

[13] J. Gray. A conversation with Werner Vogels. *ACM Queue*, 4(4):14–22, 2006.

[14] J-M. Horcas, M. Pinto, and L. Fuentes. Software product line engineering. In *Proceedings of the International Systems and Software Product Line Conference (SPLC)*, New York, NY, USA, 2019. ACM Press.

[15] R. Kazman, J. Asundi, and M. Klein. Making Architecture Design Decisions: An Economic Approach. Technical report, Carnegie Mellon Software Engineering Institute, 2002.

[16] R. Kazman, M. Klein, and P. Clements. Method for Architecture Evaluation. Technical report, Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, 2000.

[17] S. Newman. *Monolith to Microservices: Evolutionary Patterns to Transform Your Monolith*. O'Reilly Media, 2019.

[18] D. L. Parnas. On the criteria to be used in decomposing systems into modules. *Communications of the ACM*, 15(12):1053–1058, dec 1972.

[19] E. R. Poort and H. Van Vliet. RCDA: Architecting as a risk- and cost management discipline. *Journal of Systems and Software*, 85(9):1995–2013, 2012.

[20] P. Runeson and M. Höst. Guidelines for conducting and reporting case study research in software engineering. *Empirical Software Engineering*, 14(2):131–164, 2009.

[21] Swedish Standards Institute. SIS-ISO/IEC TR 9126-2:2003 Software engineering – Product quality – Part 2: External metrics. Technical Report 121124, 2003.

[22] D. Taibi, V. Lenarduzzi, and C. Pahl. Processes, Motivations, and Issues for Migrating to Microservices Architectures: An Empirical Investigation. *IEEE Cloud Computing*, 4(5):22–32, sep 2017.

Paper C.
Towards a More Reliable Store-and-forward Protocol for Mobile Text Messages

Towards a More Reliable Store-and-forward Protocol for Mobile Text Messages. Daniel Brahneborg, Wasif Afzal, Adnan Čaušević, and Mats Björkman.
In *Proceedings of the Workshop on Advanced tools, programming languages, and PLatforms for Implementing and Evaluating algorithms for Distributed systems (ApPLIED), part of the Principles Of Distributed Computing conference (PODC)*. ACM, 2018.

Abstract

Businesses often use mobile text messages (SMS) as a cost effective and universal way of communicating concise information to their customers. Today, these messages are usually sent via SMS brokers, which forward them further to the next stakeholder, typically the various mobile operators, and then the messages eventually reach the intended recipients. Infoflex Connect AB delivers an SMS gateway application to the brokers with the main responsibility of reliable message delivery within set quality thresholds. However, the protocols used for SMS communication are not designed for reliability and thus messages may be lost.

In this position paper we deduce requirements for a new protocol for routing messages through the SMS gateway application running at a set of broker nodes, in order to increase the reliability. The requirements cover important topics for the required communication protocol such as event ordering, message handling and system membership. The specification of such requirements sets the foundation for the forthcoming design and implementation of such a protocol and its evaluation.

7.1 Introduction

Today, mobile text messages (a.k.a. SMS) are often used in business-to-consumer communication, e.g., two-factor authentication and booking reminders. Text messages provide quick and cost effective communication with world-wide coverage, making them a natural choice in many situations. However, the concrete implementation for processing these messages reveals a rather complex system. There are many mobile network operators, even within the same country, and each message must be sent to the correct operator in order to reach the intended recipient. In the 1990s it was sufficient to examine the first few digits in the destination number to find the correct operator. That is no longer enough, as number portability allows customers to keep their number while switching operators. Additionally, the operators use a plethora of communication protocols (e.g. SMPP, UCP, CIMD2, HTTP), of different versions (SMPP 3.4 allows bi-directional traffic while SMPP 3.3 does not) or with unique implementation issues and requirements (primarily phone number formats and character encodings).

Businesses commonly send text messages via services provided by SMS brokers instead of handling this complexity themselves. These brokers charge a fee for providing a single protocol to their clients (i.e. the businesses), and handle both technical and financial communications with the operators. Our context is the software used by these SMS brokers to forward the text messages from their clients to different operators. We call this the SMS gateway application. One such application is the Enterprise Messaging Gateway (EMG) from Infoflex Connect AB.

SMS messaging is based on a store-and-forward architecture, similarly to TCP/IP. Incoming messages are stored in a local queue, and from that queue they are extracted and forwarded as soon as possible. Figure 7.10 shows the data flow for a message sent from the client to a node run by an SMS broker, and then forwarded to a mobile operator. When the client receives the response at point D, the SMS broker has assumed full responsibility for the message, so the client will delete their copy of the message. A similar response is sent at point G. When the recipient finally has received the message from the operator, an acknowledgement is returned to the sender, indicating whether the message needs to be resent. For TCP/IP that acknowledgement is an ACK packet, and for SMS it is a delivery report (usually shortened to "DLR"), albeit with an important difference: the DLR is unreliable. The unreliability of the delivery report is our main issue, as even though the sender can use the response from the SMS broker or operator to know whether the message was accepted, they can not use the DLR to know if the message reached its final destination.

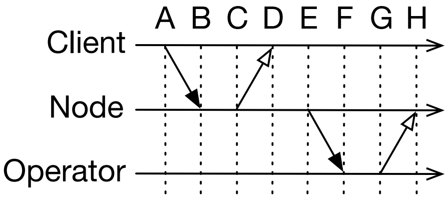


Figure 7.10: Traffic between clients, nodes, and operators. Filled arrows represent messages, and hollow arrows represent responses.

The critical section here is between points C and E in Figure 7.10, when the message has been received and acknowledged, but not yet forwarded. If the node were to crash in this interval, the message would be lost. To mitigate this risk, the message must be replicated to one or more additional nodes before the response is sent at C. The connectivity to the operator may be (temporarily) broken or too slow in relation to the rate of incoming traffic, resulting in thousands of messages stored on a node, waiting to be forwarded. Without replication, these messages could be lost.

At Infoflex Connect AB, we have experimented with storing messages in SQL and NoSQL databases, but results were discouraging from a performance perspective. Typical throughput was at most a few hundred messages per second, while we could achieve ten times that when using an embedded database, and hundred times that when storing the messages only in memory. There are several reasons for this, apart from the additional I/O. For a system with a single node, the message queues are kept in memory in self-balancing sorted trees to keep the messages in order. For persistence, messages are written and removed using their unique identifier. In a system with multiple nodes, the ordering must be handled by the database, which is not only costly in itself, but also requires synchronization between the database nodes, which in turn must be done for every message.

In this position paper we discuss the requirements for, and the effects of, adding a replication protocol in this store-and-forward context. We do this by first describing our system model (Section 7.2), followed by the requirements to be met by the protocol in our context (Section 7.3). Solution candidates are discussed in Section 7.4, while Section 7.5 contains a more general review of related work. Section 7.6 summarizes the requirements and the considered approaches, and concludes the paper.

7.2 System Model

We assume the perspective of an SMS broker, using a system consisting of a collection of nodes, some of which are geographically distant. Each node has a unique and constant identifier, can connect to all other nodes via Internet, and may join and leave the system at any time. Furthermore, the nodes are crash-recovery, so they may rejoin after crashing.

Each node runs a store-and-forward application in a configuration as shown in Figure 7.11. Messages are sent by clients, stored in local queues on one of the nodes managed by an SMS broker, and then forwarded to one of the operators after which they are removed from the queue.

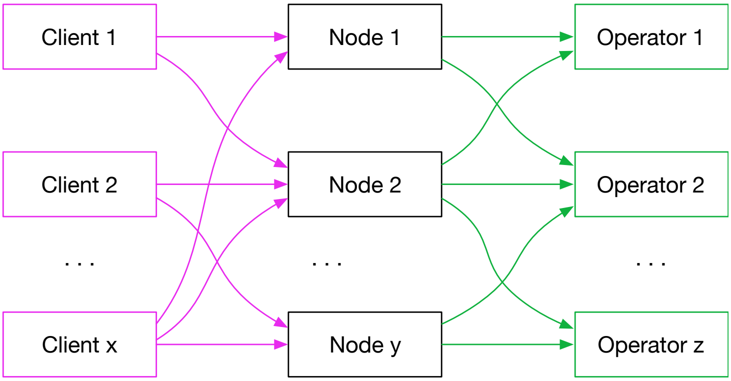


Figure 7.11: Clients, nodes, and operators.

Security issues such as authentication and encryption are handled separately, and there are no byzantine failures [19] with nodes sending arbitrarily erroneous data.

7.3 Requirements

We group our requirements into the following categories: 1) ordering, 2) message handling, 3) system membership, 4) metadata handling, 5) message ownership, and 6) third party effects. Each one is described next. All requirements are considered critical except for the moving of messages between nodes described in Section 7.3.5.2, as this is just a performance optimization.

7.3.1 Ordering

The most important requirement in this context, as it has the most far-reaching effects on the solution space, and to the best of our knowledge is the most novel one, is an anti-requirement: the order in which messages are forwarded does not matter. Considering the uses of mobile text message, it is easy to see why. If, e.g., two users request a new authentication code within a few seconds of each other, the exact order of delivery of the codes to the users' mobile phones is not important. For the same reason, the global order of messages received on different nodes, is not important either [24]. However, for the sake of fairness and to ensure liveness, messages should be forwarded in approximately the same order as they were received.

The most commonly used protocols for SMS communication, e.g., SMPP and UCP, support sliding windows with transaction numbers. These numbers are unique values set by the sender and duplicated in the responses, making the reception order of the responses irrelevant. Combined with the insignificance of the message ordering, the protocol can decide to reorder events in different ways if necessary.

7.3.2 Message handling

Several of the requirements relate to replicating incoming messages and forwarding the messages to an operator or another SMS broker.

7.3.2.1 Replicate incoming messages

The first step towards high reliability is ensuring the incoming client messages are replicated to the other nodes. The set of nodes required to confirm receiving the messages before the response is sent constitute a quorum [11], which for n nodes in a normal majority system needs a size of at least $\lfloor n/2 \rfloor + 1$. Other quorum systems [34] use other sizes. Thus, we assume this quorum size to be configurable.

7.3.2.2 Forward messages

Each message received by a node should, ideally, be forwarded to an operator exactly once, regardless of the number of nodes in the system and how many nodes the message has been replicated to. However, this "exactly once" requirement is not absolute, as it is sometimes violated by the usage of sliding windows mentioned in Section 7.3.1. Sliding windows can lead to duplicated messages if the connection breaks after the message has been received by the operator but before the response comes back to the node (i.e. the interval between points G and H in Figure 7.10).

In a typical configuration with a single node, the sender uses a window size, denoted w, between 1 and 10. When using UCP, w is limited to 99. With a window size of w, they can send w messages before requiring a response, so the number of messages with an unknown status in case of a broken connection is at most w. The maximum number of duplicated messages, per broken connection, is therefore w. In a system with multiple nodes, the number of duplicated messages should still not exceed w. This would be possible to verify by using a model checker, e.g. Spin[2] or Uppaal[3].

7.3.3 System membership

The set of nodes in the system should be able to both grow and shrink dynamically.

7.3.3.1 Accept a new or returning node to the system

A node should be able to join the system at any time, simply by connecting to one of the existing nodes. As described in our system model in Section 7.2, the nodes are crash-recovery, meaning that nodes can reconnect to the rest of the system, in particular if they were just temporarily unavailable due to a network partition. Even a short-lived network partition may last longer than the lifetime of the messages in the queues, so the difference between "new node" and "returning node" is expected to be minuscule.

7.3.3.2 Remove a node from the system

We want the current set of nodes in the system to be known to all nodes as soon as possible, in order to know which nodes to replicate messages to. In case the application must be manually stopped,[4] the protocol should propagate the information about a node's impending death.

7.3.4 Metadata handling

For financial reasons, we must keep track of all received and forwarded messages. We do this by updating the client's credit, and keeping the current state of all messages in a global database.

[2] `http://spinroot.com`

[3] `http://www.uppaal.org`

[4] There are many possible reasons for this, e.g., it should be replaced with a newer version or its system configuration may have changed in a way that requires a full restart.

7.3.4.1 Manage Credits

Before the client is allowed to send a message, the client's current credit value should be examined. A possible overdraft of this credit is acceptable if it lowers the round-trip time and increases the throughput, but this overdraft must be limited and configurable. When the real cost of forwarding the message is known, this credit value is updated.

7.3.4.2 Message State Database

For audit purposes, there must exist a mechanism for retrieving the state of all received and forwarded messages. This information does not have to be exact at every point in time, as long as each message is only counted once. Therefore, eventual consistency is sufficient.

7.3.5 Message Ownership

Each message can only have a single node as its owner, so when the message is replicated to the other nodes it should be stored there in a dormant state, preventing it from being forwarded by those other nodes. This replication is only useful if there also exists a mechanism for changing the owner, making it possible for the replicated messages to be forwarded by another node than the one which received them. Two of the situations where the protocol requires this mechanism are described next.

7.3.5.1 Adopt messages from a presumed dead node

When an eventually perfect failure detector reports a failed node, the other nodes should quickly take ownership of any messages currently in the queue of this node, so its messages can be forwarded. The delivery requirement is still not "exactly once", but the duplication rate should not change significantly.

7.3.5.2 Move Messages Between Nodes If Required

For applications such as SMS voting, the message can be sent "upstream" from an operator via one of our nodes, destined for one of the clients. The arrows in Figure 7.11 just indicates who is connecting to who, the actual network traffic is bidirectional. If the client is not connected to all nodes, a mechanism is needed to automatically move messages to one of the nodes where the client is connected.

Referring to Figure 7.12, consider the case when Operator z has a message destined for Client 1. The operator does not know about the connections on the

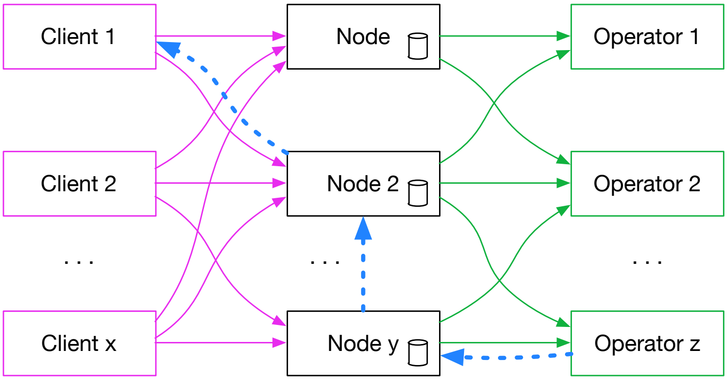


Figure 7.12: Routing a message back to the client.

left side, so it is sent to a randomly selected node, which in this case could be Node y. It would have been better to send it to Node 2, as it could then be sent directly to Client 1, but we have no control over that. Client 1 is not connected to Node y, so the message must be moved to Node 1 or Node 2 before it can be forwarded to the client. In this scenario the message would need to travel according to the dashed blue lines. While certainly useful, we consider this requirement to be of low priority.

7.3.6 Third Party Effects

We must also consider the perspective of the clients, operators and SMS brokers connected to our system.

7.3.6.1 Transparent to third party software

The communication with both clients and operators follow well defined protocols (e.g. SMPP or HTTP), involving thousands of clients and hundreds of operators. Any solution must therefore be completely transparent to these third parties. However, we can assume clients can be given the connection details to multiple nodes and that they can switch freely between them, in particular if the selected node becomes unreachable. HTTP includes response codes to request such a switch, but we can not depend on that being supported by the clients.

As more work needs to be performed per message in order to perform the replication, and additional round-trips between the nodes must be performed before the client can get their response back, the round-trip time seen from the client's perspective (Figure 7.10, the interval between points A and D) will increase with any replication strategy.

7.3.6.2 High throughput

The usage of local queues gives mostly independent nodes, which should allow the throughput to remain high while still getting increased reliability from the replication. Previous experiments using a shared database have failed to achieve more than a few hundred messages per second even within the same data center, while local queues reach several thousand. The interval between 1000 and 10 000 operations per second is what is achieved by WanKeeper [1] when it focuses on reading data, as it can take advantage of local processing and data ownership. With equal parts reading and writing in a configuration with geographically distant servers [1], WanKeeper achieves 100 operations per second, while the commonly used ZooKeeper achieves 10. Our requirement, based on discussions with various SMS brokers, is 1000 or more operations per second.

7.3.7 Limitations

A system fulfilling the described requirements would have some limitations we need to be aware of.

System reliability & performance evaluation

To verify the reliability and performance of the solution, extensive testing and verification is needed.

Network overhead

The replication will lead to increased network traffic, but we can mitigate that in several ways. First, we can replicate multiple messages in the same packet, using the network bandwidth as effectively as possible. Second, we can replicate the messages to just a subset of the nodes. Third, messages that have been forwarded before being replicated to the most distant nodes (with the longest round-trip time), does not have to replicated there at all.

Increased complexity

The complexity of the system would increase, primarily caused by the coordination between the nodes. This could to be addressed by making the solution architecture as simple as possible.

Protocol adoption

Despite being called a protocol, the requirements actually concern an internal architecture. It is not intended for interoperability between separate systems, rather between different broker nodes within a single system.

7.4 Solution Space

This section discusses various available solutions to address the elicited requirements as well as their suitability in our context.

7.4.1 SQL and NoSQL database clusters

In a multi-node environment a clustered database might be considered for managing the credit values, as described in Section 7.3.4.1.

Section 7.3.2 described the low consistency requirements in this domain, which is lower than what either ACID (Atomicity, Consistency, Isolation, and Durability) [12] or BASE (Basically Available, Soft state, Eventual consistency) [10] can offer. Similarly to how an optimistic consistency model can lead to better performance [31], this relaxed delivery guarantee should also enable a more effective architecture.

7.4.2 Adoption tokens

Our initial idea for solving the changes of message ownership described in Section 7.3.5 was to use an "adoption token", and passing it around between the nodes. Only the node currently in possession of this token would be allowed to adopt any messages, so there should never exist more than one. If a node dies or there is a network partition, the system may end up with both 0, 1 or 2 such tokens. A lost token is no problem, as that situation is no different from when the system is initially started; simply run a leader election algorithm such as Paxos [18] or Raft [26] to select a node that can create the token. Multiple tokens on the other hand, could lead to thousands of duplicated messages. This can happen if the adoption token ends up in the minority group during a network partition.

Next, we kept the idea of an adoption token, but modified the token passing to use quorum voting instead. To pass the token, the node currently holding it would start an election among all reachable nodes, suggesting the next node. When the new node has seen enough votes, the token is recreated there.

However, this can also lead to considerable message duplications. Consider the following sequence of events, in a system of 3 nodes: A, B and C.

1. Node A has the token, and starts an election for passing it to B.
2. Node B wins the election, and assumes ownership of the token.
3. The network splits, leaving B alone.
4. Node B realizes it can not reach node A, and adopts all its messages. At this point all current messages on node A are duplicated. Node A is still in the majority group, so it keeps processing its messages normally.
5. Node C realizes it has not seen the adoption token in some time, and starts an election to create one.
6. Node C gets a vote from A, giving it a majority.
7. Node C realizes it can not reach node B, and adopts all its messages. Now all messages on node B are also duplicated.

Depending on the relative executions of the nodes, we may still end up with two adoption tokens and thousands of duplicated messages as an unacceptable consequence.

7.4.3 Replicated logs

In Section 7.3.2.2 we concluded that the messages have no linearizability requirement. The high independence between separate text messages, combined with the throughput requirement, make solutions that send all events via a single node to ensure all nodes see the events in the same order both unnecessarily strict and unusable. For this reason, we are not able to use neither Raft [26], Viewstamped Replication [20], nor ZooKeeper [14] for the external network traffic.

The externally visible protocols can not be changed, as discussed in Section 7.3.6.1. This prevents solutions such as Raft [26] and Chubby [4].

7.5 Related Work

This section discusses the existing products and academic work focused on data replication. In Section 7.5.1 we describe solutions that are implemented, evaluated and in use, while in Section 7.5.2 we focus more on theoretical results. In short, we have not been able to find a solution that can take advantage of our low ordering requirements, thus reusing existing solutions would potentially result in suboptimal performance in our context.

7.5.1 State of practice

In many applications, persisting data to survive application crashes is done in a database. This can be either a classical SQL database such as MySQL or Oracle, or a more modern NoSQL database such as MongoDB or Cassandra.

For message queues, RabbitMQ and Apache Kafka are common solutions. Kafka is a better choice if events need to be persisted to disk and replayed any number of times by clients, while RabbitMQ supports multiple protocols which is good for interoperability, and a possibility to set the time to live (TTL) of messages [5]. However, the disk usage by Kafka can be extensive, and RabbitMQ does not scale well when the queue sizes increase. EMG needs both support for message TTL and large queues.

7.5.1.1 Message Queues

There are plenty of message queue products of varying complexity (e.g. RabbitMQ[5], Qpid[6], and IBM WebSphere), implementing various well documented protocols (e.g. Java JMS, AMQP (Advanced Message Queuing Protocol)[7], and ZeroMQ [13]), so a program that only forwards messages is trivial. However, systems using these solutions, such as SDQS [38], Andes [35] and EQS [32], are all very strict regarding the ordering of the messages. A common solution for synchronization between nodes to ensure this ordering is ZooKeeper [14], which works fine for local clusters within a single data center, but in a geo-distributed environment stays below 10 transactions per second [1].

Previously, the message queue system Apache Kafka[8] [15] used ZooKeeper [14] for coordination for each entry, making it unusable in our context. In Kafka, enqueued messages can be delivered to one of many nodes, which is exactly what we needed. Messages can be separated into different topics, with ordering only being guaranteed within each topic. This relaxed ordering is still too strict for us.

Closely related to message queues are publish/subscribe systems. Both message queues and publish/subscribe systems allow “space decoupling”, i.e. the sender and the recipient need not be aware of each other, “time decoupling”, i.e. the message is sent at one point in time and delivered at another,

[5] `https://www.rabbitmq.com`

[6] `https://qpid.apache.org`

[7] `http://www.amqp.org`

[8] `http://kafka.apache.org`

and “synchronization decoupling”, i.e. the sender does not have to wait until the message has been delivered to the recipient [8]. This matches our requirements well. The two main ways they differ from what is needed for EMG is that they also assume a “one-to-many” delivery strategy, and that filtering must be used to avoid sending all messages to all recipients.

7.5.1.2 Storage

An important difference between our message queues and a generic storage system, is that we have no externally initiated reads. Once a message has been received, only the system itself needs to know where it is stored. There will be no requests from other applications to fetch the message. Therefore, all effort spent in handling such operations, are for our purposes wasted. As an example, the “Saturn” system [3] is described as a way to “enforce causal consistency when accessing replicated data”, where the critical word here is “accessing”. Other systems such as ChainReaction [2], Orbe [6], GentleRain [7], COPS [22] and SwiftCloud [37] use different mechanisms to achieve this accessibility, e.g. vector clocks [17], physical clocks, and caches. MeteorShower [21] explicitly addresses the delays caused by geographically separated servers.

Rarely, if ever, is the lifetime of the stored objects addressed. Typically, objects are created by one of the nodes, and occasionally updated by the same or another node. Most of the operations are then read accesses. For example, the system OCC [31] uses a workload “with a 32:1 GET:PUT ratio”. For a message queue, the situation is different. For such a system, there are not only no read accesses at all, we also know that all objects will be removed after usually just a few seconds. To the best of our knowledge, no existing storage mechanism considers this factor as a way of minimizing the amount of data needed for the replication.

7.5.2 State of the art

The most active areas of interest to us concern commutative functions and leader election algorithms. Commutative functions address the fact that strict ordering of events is unnecessary, and leader election is the base concept when achieving consensus between multiple nodes.

7.5.2.1 Commutative Functions

Whether reality is ordered has been discussed for some time, at least from 1993 by Queinnec and Padiou [29] regarding flight plans. If this ordering can be ignored, it is also possible to attain higher availability of data in an unreliable network [9] and lower the number of network round-trips [27]. For example, using the CRDT "PN-counter" [30], it is possible to implement updates and limit checks of user credits in a distributed environment from Section 7.3.4.1, without any network round-trips.

Using commutative functions in order to get lower response times has been known since 1988 [16] if not earlier. Shapiro et al. [30] described data types based on these commutative functions, calling them "Convergent or Commutative Replicated Data Types (CRDTs)". Zawirski then described some lower limits of the amount of metadata needed for the replication of some of these data types [36], both in terms of the number of nodes and the number of updates.

7.5.2.2 Leader Election

There are several algorithms for reaching consensus among a set of communicating nodes [33], each one in multiple variants. The most commonly known one is probably Paxos [18]. In recent years, Raft [26] is also used. Both of them, as well as Viewstamped Replication [20, 25] and the lock service Chubby [4], require support from the third party applications that connect to the replicating system. This is in conflict with the requirement in Section 7.3.6.1 of being transparent to those third parties. For an internal protocol between EMG nodes only, any of these can of course be used.

To avoid overloading a single node with a leadership role through which all requests much pass, the mechanism used by Paxos, Raft and others, some alternatives exist. Mencius [23] is derived from Paxos, but lets the leader role rotate periodically. AllConcur [28] goes further, being entirely leaderless. With AllConcur, each message is forwarded several times between different pairs of nodes, resulting in more network traffic than the leader-based methods.

7.6 Summary

7.6.1 Approaches

Table 7.3 summarizes the requirements and approaches to be considered for fulfilling them. Essentially, databases are too strict on ordering, and replicated logs are unsuitable in a geo-distributed environment as they rely on all events being serialized by a single node.

[9] Information about new nodes are broadcast among all nodes, but each node maintains its own list of active peers.

Requirement	**Approach**	**Problems**
System membership	Per node[9]	None known
Message storage	Database	Strict ordering
	Message queue	Strict ordering
Message state	Database	Round-trip times
	Replicated log	Single node
Message ownership	Replicated log	Single node
Client credits	PN-counter	Possible overdrafts

Table 7.3: Considered approaches for each set of requirements, and expected new problems.

7.6.2 Conclusions

The communication protocols for mobile text messages are unreliable, as there is no dependable end-to-end acknowledgement packet. There is, on the other hand, no strict ordering requirements in this domain, allowing the use of more effective solutions than off-the-shelf message queue products. We can not avoid the inevitable round-trip time between data centers, which may very well be geographically distant, but by replicating messages from multiple clients in the same update, the total system throughput should be satisfactory. As contributions, this position paper sets the requirements for a reliable communication protocol for SMS in place, and reviews the state of the art as well as practice for considered solutions. While partial solutions suitable in our context are available, a complete solution satisfying the requirements specified in this paper would require a new bespoke protocol that can effectively take advantage of the possibility of event reordering and short lifetime of the messages.

Acknowledgments

This work was sponsored by The Knowledge Foundation industrial PhD school ITS ESS-H, 20160139 (TestMine), 20130085 (TOCSYC) and Infoflex Connect AB.

Bibliography

[1] A. Ailijiang, A. Charapko, M. Demirbas, B. O. Turkkan, and T. Kosar. Efficient distributed coordination at WAN-scale. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017.

[2] S. Almeida, J. Leitão, and L. Rodrigues. ChainReaction: a Causal+ Consistent Datastore based on Chain Replication. In *Proceedings of The European Professional Society on Computer Systems (EuroSys)*. ACM, 2013.

[3] M. Bravo, L. Rodrigues, and P. Van Roy. Towards a Scalable, Distributed Metadata Service for Causal Consistency under Partial Geo-replication. *Proceedings of the Doctoral Symposium of the International Middleware Conference (Middleware)*, 2015.

[4] M. Burrows. The Chubby lock service for loosely-coupled distributed systems. In *Proceedings of the Symposium on Operating Systems Design and Implementation*, OSDI. USENIX Association, 2006.

[5] P. Dobbelaere and K. S. Esmaili. Kafka versus RabbitMQ. In *Proceedings of the Conference on Distributed Event-Based Systems (DEBS)*. ACM, 2017.

[6] J. Du, S. Elnikety, A. Roy, and W. Zwaenepoel. Orbe: Scalable causal consistency using dependency matrices and physical clocks. In *Proceedings of the Symposium on Cloud Computing (SOCC)*. ACM, 2013.

[7] J. Du, C. Iorgulescu, A. Roy, and W. Zwaenepoel. GentleRain: Cheap and Scalable Causal Consistency with Physical Clocks. *Proceedings of the Symposium on Cloud Computing (SOCC)*, 2014.

[8] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[9] M. J. Fischer and A. Michael. Sacrificing Serializability to Attain High Availability of Data in an Unreliable Network. In *Proceedings of the SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS)*. ACM, 1982.

[10] A. Fox, S. D. Gribble, Y. Chawathe, E. A. Brewer, and P. Gauthier. Cluster-based scalable network services. *ACM SIGOPS Operating Systems Review*, 31(5), 1997.

[11] D. K. Gifford. Weighted Voting for Replicated Data. In *Proceedings of the Symposium on Operating Systems Principles (SOSP)*. ACM, 1979.

[12] T. Haerder and A. Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys*, 15(4):287–317, 1983.

[13] P. Hintjens. *ZeroMQ: messaging for many applications*. O'Reilly Media, Inc., 2013.

[14] P. Hunt, M. Konar, F. P. Junqueira, and B. Reed. ZooKeeper: Wait-free Coordination for Internet-scale Systems. In *Proceedings of the USENIX Annual Technical Conference (USENIX ATC)*. USENIX Association, 2010.

[15] J. Kreps, N. Narkhede, and J. Rao. Kafka: a Distributed Messaging System for Log Processing. In *Proceedings of the SIGMOD Workshop on Networking Meets Databases (NetDB)*, Athens, Greece, 2011.

[16] A. Kumar and M. Stonebraker. Semantics Based Transaction Management Techniques for Replicated Data. *Proceedings of the SIGMOD International Conference on Management of Data (SIGMOD)*, 1988.

[17] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, 1978.

[18] L. Lamport. The Part-time Parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.

[19] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3), 1982.

[20] B. Liskov and J. Cowling. Viewstamped Replication Revisited. Technical Report MIT-CSAIL-TR-2012-021, Massachusetts Institute of Technology, 2012.

[21] Y. Liu, X. Guan, V. Vlassov, and S. Haridi. MeteorShower: Minimizing Request Latency for Majority Quorum-Based Data Consistency Algorithms in Multiple Data Centers. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017.

[22] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. Don't Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS. *Proceedings of the Symposium on Operating Systems Principles (SOPS)*, 2011.

[23] Y. Mao, F. P. Junqueira, and K. Marzullo. Mencius: Building Efficient Replicated State Machines for WANs. In *Proceedings of the USENIX Conference Operating System Design and Implementation (OSDI)*, 2008.

[24] C. S. Meiklejohn. A certain tendency of the database community. In *Companion to the First International Conference on the Art, Science and Engineering of Programming*, Programming '17, pages 34:1–34:5, New York, NY, USA, 2017. ACM.

[25] B. M. Oki and B. H. Liskov. Viewstamped Replication: A New Primary Copy Method to Support Highly-Available Distributed Systems. In *Proceedings of the Symposium on Principles of Distributed Computing (PODC)*. ACM, 1988.

[26] D. Ongaro. *Consensus: Bridging Theory And Practice*. PhD thesis, Stanford University, 2014.

[27] S. J. Park and J. Ousterhout. Exploiting Commutativity For Practical Fast Replication. 2017.

[28] M. Poke, T. Hoefler, and C. W. Glass. AllConcur: Leaderless Concurrent Atomic Broadcast Marius. In *Proceedings of the International Symposium on High-Performance Parallel and Distributed Computing (HPDC)*. ACM, 2017.

[29] P. Queinnec and G. Padiou. Flight plan management in a distributed air traffic control system. *Proceedings of the International Symposium on Autonomous Decentralized Systems (ISAD)*, 1993.

[30] M. Shapiro, N. Pregui, C. Baquero, and M. Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types. Technical Report RR-7506, Inria – Centre Paris-Rocquencourt, 2011.

[31] K. Spirovska, D. Didona, and W. Zwaenepoel. Optimistic Causal Consistency for Geo-Replicated Key-Value Stores. In *Proceedings of the International Conference on Distributed Computing Systems (ICDCS)*. IEEE, 2017.

[32] N. L. Tran, S. Skhiri, and E. Zimányi. EQS: An elastic and scalable message queue for the cloud. In *Proceedings of the International Conference on Cloud Computing Technology and Science, (CloudCom)*. IEEE, 2011.

[33] J. Turek and D. Shasha. The Many Faces of Consensus in Distributed Systems. *Computer*, 25(6):8–17, 1992.

[34] M. Vukolić. *Quorum Systems: With Applications to Storage and Consensus*. Synthesis Lectures on Distributed Computing Theory. Morgan & Claypool Publishers, 2012.

[35] C. Wickramarachchi, S. Perera, S. Jayasinghe, and S. Weerawarana. Andes: A highly scalable persistent messaging system. In *Proceedings of the International Conference on Web Services (ICWS)*. IEEE, 2012.

[36] M. Zawirski. *Dependable Eventual Consistency with Replicated Data Types*. PhD thesis, Université Pierre et Marie Curie - Paris, 2015.

[37] M. Zawirski, N. Preguiça, S. Duarte, A. Bieniusa, V. Balegas, and M. Shapiro. Write Fast, Read in the Past: Casual Consistency for Client-side Applications. Technical report, 2015.

[38] Z. Zhang, Y. Wang, H. Chen, M. Kim, J. M. Xu, and H. Lei. A cloud queuing service with strong consistency and high availability. *IBM Journal of Research and Development*, 55(6):10:1–10:12, 2011.

Paper D.
GeoRep – Resilient Storage for Wide Area Networks

GeoRep – Resilient Storage for Wide Area Networks. Daniel Brahneborg, Romaric Duvignau, Wasif Afzal, Saad Mubeen.
In *IEEE Access*, 2022.

Abstract

Embedded systems typically have limited processing and storage capabilities, and may only intermittently be powered on. After sending data from its sensors upstream, the system must therefore be able to trust that the data, once acknowledged, is not lost. The purpose of this work is to propose a novel solution for replicating data between the upstream nodes in such systems, with a minimal effect on the software architecture. On the assumption that there is no relative order between replicated data tuples, we designed a new replication protocol based on partial replication. Our protocol uses only 2 communication steps per data tuple, instead of the 3 to 12 used by other solutions. We verified its failover mechanism in a proof-of-concept implementation of the protocol using simulated network failures, and evaluated the implementation on throughput and latency in several controlled experiments using up to 7 nodes in up to 5 geographically separated areas, with up to 1000 data producers per node. The recorded system throughput increased linearly relative to both the number of nodes and the number of data producers. For comparison, Paxos showed a performance similar to our protocol when using 3 nodes, but got slower as more nodes were added. The lack of a relative order, in combination with partial replication, enables our system to continue working during network partitions, not only in the part containing the majority of the nodes, but also in any sufficiently large minority partitions.

8.1 Introduction

All over the world, various types of disasters happen with both regular and irregular intervals [13, 37, 48, 49]. These disasters, which could be caused by natural, technical, political or other kinds of events, affect network and power equipment, and might therefore lead to outages for internet services [1, 8, 63]. Such infrastructure failures have been showed to be about twice as likely the cause for services being unavailable to clients, as compared to failures in the servers themselves [18]. Oftentimes, these infrastructure failures can be mitigated by using multiple geographically separated servers [13, 52, 53, 62], conveniently offering protection from failures in both infrastructure and individual servers. The servers exchange data with each other as necessary, allowing clients to connect to any one of them. If the system uses different cloud providers for each data center to mitigate the risk of failures due to software or configuration upgrades [26], the probability for some event killing multiple nodes during the processing of a particular data tuple is effectively zero.

Maintaining the same data on multiple servers is not a new problem. A common solution is to use *full replication*, which sends all information regarding the processed data to all other servers [11]. This is often managed via a master server as in Paxos [35, 43] or Raft [50], ensuring both that all data and its operations are communicated to all servers, and that the operations are processed in the same order [6].

Full replication is easy to understand and reason about, and is implemented in various concrete tools and libraries, e.g., Redis[10] and Spread.[11] It forms the basis for eventual consistency [60], and for Convergent and Commutative Replicated Data Types (CRDTs) [57]. It is good for web applications and other request-response based systems as it gives good availability for external readers, which can send the requests to any one of the included servers and get reasonably current data in return. Because the system can freely select one or more remaining servers to take over the duties of a failed server [40], this also makes resilience, as described by the ResiliNets project [48, 58], straightforward. Resilience is then the degree of how well a system can recover from failures. This differs from robustness, which is how well the system behaves during normal operations.

However, full replication also has a number of shortcomings. It wastes network traffic [36, 37], as the amount of transmitted data grows at least linearly by the number of servers in the system. It requires all servers to be able to reach each other, possibly going via one or more other servers. When there is

[10] `https://redis.io`

[11] `http://www.spread.org`

a network partition, by which we mean any type of failure breaking full reachability, system availability [5, 19, 23, 31, 53] is reduced as clients can then only perform updates on the nodes in the remaining majority part, if any. The required coordination can be costly [29, 34] and limit system performance.

In this work, we envision an application providing a message queue for event data sent from sophisticated sensors or IoT devices. The data tuples are added to the queue by the devices, and then one by one pushed by the queue itself to the service responsible for that particular type of data. After being successfully forwarded, each data tuple is removed from the queue.

The queue's push construct has a few important implications, making previous state-of-the-art non-optimal. One of the explicit goals in current work on replication is that the data tuples should be delivered and thereby be visible to all other nodes. An alternative to this full replication is to use the more resource conservative partial replication, which only sends data tuples to a subset of the servers [53]. In our use case, each message needs to be visible to just one single server, to ensure that it is delivered only once. It is not until a server fails that the application layer on the other servers should be made aware of its messages, again only on a single server per message.

As each data tuple is independent, we have no need for consistent operation ordering, and therefore do not need any mechanism for enforcing this order [59]. As there are no external readers pulling messages from the queue, we also do not need all nodes to receive the same set of data and its operations, and thus have no use for the consistency guarantees provided by full replication.

Partial replication saves both network and other resources compared to full replication, but makes it difficult to maintain a consistent, global order between data tuples. Previous works in this area [9, 12, 14, 24, 56] solve this by using some variant of atomic broadcast [2, 15, 27, 28, 55]. Unfortunately that solution requires additional network traffic (between 1 and 10 communication steps, depending on the protocol) and relatively complex algorithms. This creates a problem with scalability, which can be observed in the literature on this topic by noticing that the system throughput does not always increase when new nodes are added. The throughput typically falls relatively quickly when the number of nodes to replicate to increases. This can, for example, be seen in the evaluation of GentleRain [20], where the throughput increases significantly slower when there are more than about 10 servers.

The purpose of this work is to design a replication protocol for a resilient message queue with high efficiency, allowing disaster-resistant processing of 1000 or more messages per second (MPS) per server, with better scalability than in state-of-the-art. The resulting design was evaluated using a proof-of-concept implementation, tested on servers scattered across multiple continents. Even on servers with modest performance, we achieved up to 3440 MPS per node in the geo-diverse case, replicating each data tuple to a random other server in the world. By always using the nearest server, e.g., from New York to Toronto, we instead reached 5661 MPS per node.

We claim the following contributions in relation with this protocol.

1. A high level description of its functionality.
2. An analysis of its reliability in terms of availability, potential data loss, and potential data duplication.
3. A method to verify its failover mechanism.
4. A performance analysis on throughput, both when deployed within a local network and for a geo-distributed system configuration.
5. An open-sourced implementation.

Following this introduction is a description of the assumptions we have made about our system model, and a sample application context. Section 8.2 describes the proposed protocol. Next follows evaluations of the protocol from three different perspectives. First, Section 8.3 contains a theoretical analysis of the reliability. Then, Section 8.4 describes the verification of the failover mechanism, and finally Section 8.5 describes the setup for the experiments conducted to evaluate the behaviour in a real-world configuration, focusing on the quality attribute throughput. The results are discussed in Section 8.6, and related work in Section 8.7. Section 8.8 holds conclusions and some ideas for future work.

This paper is an extension to the previously published conference paper [10] presenting this protocol. The main differences between that version and this updated article, are Section 8.1.3 discussing our requirements, the extension of the “Duplication Analysis” subsection into a more complete Reliability Analysis in Section 8.3, the failover verification in Section 8.4, and an extended list of references.

8.1.1 System Model

Our system model is a classic store-and-forward queue [21], with external sets of producers and consumers [22]. Data tuples, described in more detail below, are received from the producers and stored in the queue. As soon as possible after they are received, each data tuple is forwarded by the queue to one of the consumers. When acknowledged by the consumer, the data tuple is removed from our system. The data tuples are therefore managed by the queue for a relatively short time period, normally less than 1 second. There are no end-to-end acknowledgements.

The part of the system we can control and manipulate in this model is just the queue itself, which comprises a collection of n nodes, named node$_1$, node$_2$, ..., node$_n$. Each node knows about all other nodes, can exchange data with any other node, and may join and leave the system at any time. The nodes are crash-recovery, so they may rejoin after crashing. The communication between the queue nodes is asynchronous.

Each producer and consumer is a third party system connected to one or more queue nodes. We assume that each producer maintains a list of addresses to multiple nodes they can use when sending their data tuples. However, we cannot change the communication protocol used with these parties, nor anything else in their systems. Due to this, a server cannot inform clients about the other servers, unless that is already part of the protocol between clients and servers.

The data tuples contain the following fields.

id A globally unique id.

payload Opaque application specific payload.

In addition to n, the number of nodes in the system, we will use f for the number of nodes which are allowed to fail at the same time *without data being lost*. The value of f is typically 1 or 2.

We use the term "majority replication" for all data replication protocols based on inequality (8.7) below. Full replication normally uses *number of nodes to send write operations to (w)* = n and *number of nodes to read data from (r)* = 1, which trivially satisfies this condition [3]. Another variant is to wait for acknowledgements from at least $\lfloor n/2 \rfloor + 1$ nodes for both write and read operations [7].

$$w + r > n \tag{8.7}$$

Security concerns such as authentication and encryption are not part of the model. There are also no byzantine failures [44], with nodes sending arbitrarily erroneous data.

8.1.2 Example Application

One of the application areas matching our system model is application-to-human messaging, e.g. an SMS gateway. Such gateways are used by SMS brokers, connecting clients via internet to mobile network operators. These clients are companies sending event data from their IoT devices, authentication codes, meeting reminders and similar information. Using SMS makes it possible to reach all customers without them having to install any additional software on their mobile phones. Fig. 8.13 shows a schematic view of this setup. In this use case, the replication would be done between multiple SMS gateways belonging to the same SMS broker, without affecting the protocols towards neither the client companies nor the operators. In our system model, the clients are the producers, and the operators are the consumers.

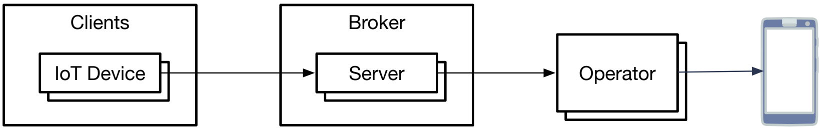


Figure 8.13: Companies sending text messages from multiple IoT devices, an SMS broker with multiple servers, mobile network operators, and customers' mobile phones.

We will use an SMS gateway for the motivation of various assumptions and decisions throughout this paper. For example, n is in this context typically at most 10. The payload field in the data tuple consists of the sender's and recipient's phone numbers, the message text, and possibly additional other information, in total a few hundred bytes.

The network operators implement their own message queues, making the mobile phone user the final consumer. This affects the delivery guarantees we must support, as it is important that all messages are delivered as soon as possible, but it is not a big problem if an occasional message is delivered twice. Similar to the established terms "at most once" and "at least once", we call this "once plus epsilon" delivery. The term "at least once" allows any number of repetitions of each message, but we want to explicitly minimize these.

8.1.3 Problem Statement and Requirements

For our store-and-forward system model in general, and our SMS application in particular, the problem addressed in this paper is to find a way to replicate the forwarded data tuples as effectively as possible, with minimal changes to an existing application. By "effectively" we mean high throughput and low CPU and network usage.

Next, we summarize our requirements, which are based on current industry standards for SMS traffic in general. An overview of the required data flows for a configuration with two nodes is shown in Fig. 8.14. A program, named

ExampleApp, is running on each node, using a context independent subsystem implementing the replication protocol. In the figure this subsystem is called GeoRep, as that is the name of our proposed solution. A producer, of which there may be many, sends data to ExampleApp on one of the nodes. The producers here correspond to the companies in Fig. 8.13. ExampleApp then tells the replication subsystem to store the data in its local persistent storage, and replicate it to the other node. When ExampleApp has forwarded the data to a consumer, corresponding to one of the operators in Fig. 8.13, it tells GeoRep to delete the data on both nodes.

The GeoRep subsystems communicate with each other for replication and failure detection. When a failed node has been detected, GeoRep tells ExampleApp on the working node to forward the data tuples originally received by the failed node. So, ExampleApp does not know anything about replication, and GeoRep knows neither of the producers nor the consumers.

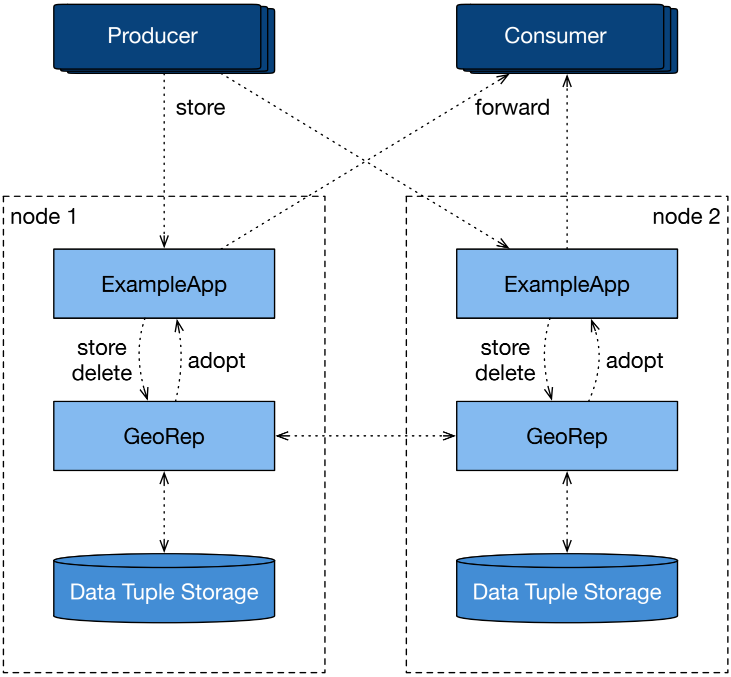


Figure 8.14: Architecture overview for ExampleApp running on two nodes.

This architecture has several advantages.

1. ExampleApp can maintain its data tuples freely, reordering and delaying them as needed, without any network traffic at all.

2. The API towards the replication system is small and generic, allowing many different solutions.
3. The replication system does not require any standalone components, which may otherwise add complexity to the installation and maintenance procedures for the full ExampleApp system.

We assume all n nodes receive the same amount of traffic, m messages per second. Using full replication will then lead to the CPU load of $\mathcal{O}(nm)$ on each node, which is undesirable as more system nodes will require a lower m. We therefore need partial replication, giving a load of $\mathcal{O}(fm)$, which is independent of n. We have set a target throughput of 1000 MPS per node.

There are a few potential solutions we need to dismiss for various reasons.

Having the "find the next data tuple" operation in the replication system
If the selection of the next data tuple to forward to the consumer is handled by the replication system, a global consensus must be reached frequently to ensure each data tuple is handled by one single node.

Apache Kafka and other standalone engines
Standalone systems have their advantages, but make the system architecture more complex as they need their own life-cycle management.

Systems requiring modifications in the producers or consumers
For example, ChainReaction [4] uses an API where new data tuples are sent to one node and acknowledged by another. Typically SMS brokers integrate with many different systems developed and maintained by other companies, making any API changes impossible in practice.

8.2 Proposed Solution

In this section we describe our proposed replication protocol, named GeoRep. It is designed to be used on n nodes, of which f nodes may fail without data being lost.

8.2.1 Protocol Description

Here we describe the activities carried out when GeoRep starts and stops, how data is replicated, and how node failures are handled.

We amend the data tuples with an additional *owners* field, containing an ordered list of $f + 1$ unique node identifiers. The first node referenced in this list is the one which originally received this tuple, and the remaining nodes are the failover nodes for this specific data tuple.

8.2.1.1 Startup

At startup, the application layer in ExampleApp provides its selected value for f to the GeoRep subsystem, and an initial list of other nodes. GeoRep then loads any previously stored data tuples into appropriate data structures in memory. When that is completed, it waits for contact requests, while also trying to make contact with the other nodes.

In response to a contact request from node$_x$, GeoRep on the contacted node returns a welcoming message with its list of currently known nodes. This list includes temporarily stopped nodes and their expected return times (see Section 8.2.1.3 below). The contacted node informs the others about node$_x$, while node$_x$ tries to connect to the existing nodes, getting their respective lists of known nodes. If any node gets an update during this phase, the full list is broadcast to all other nodes. Eventually, this will converge, from which point all nodes send periodic heartbeats [6] to all other nodes unless other data has recently been sent.

If a node returns after a short time, each welcoming message will also contain the list of entries adopted by each node. These entries can then be removed by the returning node to reduce the number of duplications.

8.2.1.2 Replication

According to our system model described in Section 8.1.1, f nodes are allowed to fail without resulting in data loss. All received data tuples must therefore be replicated to at least f additional nodes before the corresponding acknowledgement can be sent to the producer. We do not need to replicate the data to more than these f nodes, as there is no requirement of keeping all nodes identical. The replication algorithm therefore becomes as follows.

1. The application layer in ExampleApp requests some opaque data to be replicated.

2. GeoRep creates a list of f other nodes known to be alive out of the other $n - 1$ ones it knows about, putting this list in the owners field of the data tuple. If the number of alive and reachable nodes is less than f, the operation is terminated immediately, and a failure status is returned to the application. If this happens, the producer can send the data to another node.

3. The data tuple is replicated to the f selected nodes.

4. GeoRep returns a condition variable to the application. This variable is signalled when all nodes have responded. The application can therefore be as synchronous as it wants to be, while GeoRep remains asynchronous.

If multiple producers request entries to be replicated sufficiently close in time to the same node, these are all sent together. When receiving an entry from another node, it is stored locally and a response sent back, but no other action is taken. In particular, none of the received messages are forwarded at this point. Fig. 8.15 shows the replication when $n = 5$ and $f = 2$, for a message received by node$_1$, and the f other nodes being node$_3$ and node$_4$.

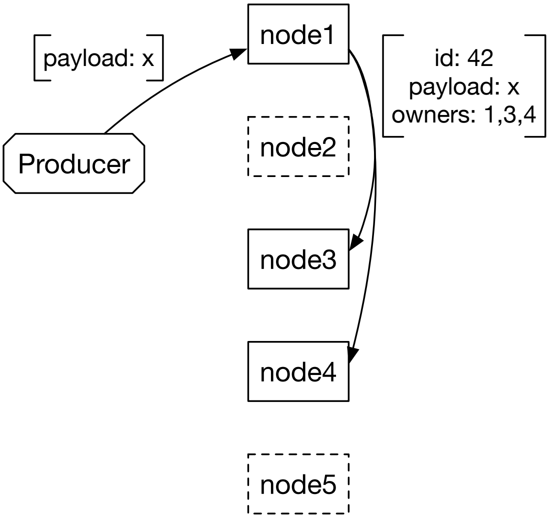


Figure 8.15: Replicate a payload to a subset of size 2 of the 5 known nodes, here nodes 3 and 4. This payload is sent neither to node2 nor node5.

8.2.1.3 Failover

If node$_1$ does not receive anything from node$_2$ for some time, node$_1$ suspects that node$_2$ is down and stops replicating entries to it [45]. It resumes replication to node$_2$ only after node$_2$ has sent proof-of-life by means of new data.

The reason for this lost connection may be a network outage, resulting in multiple isolated subsets of the original n nodes still in contact with each other. Each network partition with such a subset of at least $f + 1$ nodes can continue to run as before. This is in contrast to replication protocols using majority quorums, as they only allow the nodes in the majority to accept new data.

After some configurable time, or after the recovery timeout given by node$_2$ when it exited, node$_2$ is considered dead. If node$_1$ ends up as the first node in the owners list for one or more entries, the application running on node$_1$ is notified, one entry at a time. For these entries, node$_1$ is now the only node allowed to forward them to the consumer. We call this transfer of ownership *adoption*. The identifiers of the adopted and successfully sent entries are stored for a limited time, making it possible to notify node$_2$ should it return.

As node$_1$ knows the identifiers of the rest of the nodes to which each entry was replicated, it will try to inform those nodes about updated statuses. Only the nodes in the owners list will ever send updates and deletes for a particular entry, and only to the nodes originally stored in that list.

8.2.1.4 Exiting

When ExampleApp exits and tells GeoRep to shut down, this event is broadcast to all other nodes, including a timeout for when the node expects to be back. This timeout is also stored locally. The timeout tells the other nodes when they can start adopting that node's messages. If the original node comes back after the timeout has expired, it can assume all of its messages have been adopted by the other nodes.

8.2.2 Peer Life Cycle

Fig. 8.16 shows the states and transitions used by each node for each one of the other nodes. A node maintains its own list of states for these peer nodes, so all nodes can take different decisions on which other nodes to replicate data to. This is intentional, and an important feature of this replication protocol as it both avoids having to reach consensus on the set of reachable servers, and allows the protocol to continue to work even in case of partial failures. As our model has crash-recovery nodes, there is no end state.

When a node is informed about the existence of a new peer, the new peer starts in the *Prospect* state, causing the node to send it a greeting. When the peer replies with some data, regardless of the current state, it is moved to the *Active* state. This is the only state where it can receive new data tuples, and is marked with **boldface**.

When no data has been received for some time, the peer first moves to the state *Schrödinger*, and after an additional time to the state *Terminated*. The timeouts when moving to the *Schrödinger* and *Terminated* states are configurable, letting the application select its sensitivity to timeouts. When a node knows it will be away for just a short while, making any failover adoptions

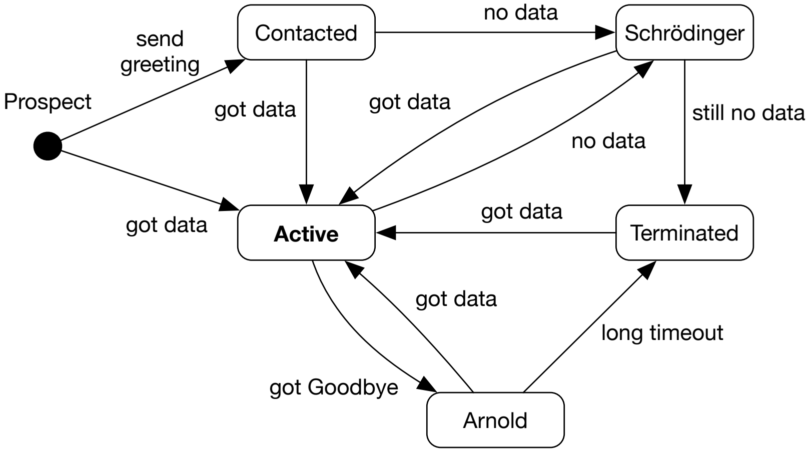


Figure 8.16: The life cycle of each peer.

unnecessary, it can send a goodbye message to the other nodes which puts it in the *Arnold*[12] state. The failover logic is triggered when moving to the *Terminated* state. To allow partitions to heal, all nodes send occasional heartbeats even to *Terminated* nodes.

8.2.3 Data Tuple Life Cycle

Fig. 8.17 and Fig. 8.18 illustrate the replication and failover from the perspective of a single data tuple. The *Inactive* state has a dashed border to show that it is a passive state, waiting on an externally initiated event. The solid arrows represent state changes on the first node, and dashed arrows on the failover nodes.

First, in Fig. 8.17, a producer sends the data tuple to some node, whereby the data tuple enters the *Received* state. This corresponds to the arrow from *Producer* to node$_1$ in Fig. 8.15. Next, this node sets the owners field, and replicates the updated data tuple to the selected failover nodes, where they are stored in the *Inactive* state. Also in Fig. 8.15, these are the arrows on the right, from node$_1$ to node$_3$ and node$_4$. When the failover nodes have confirmed this operation, the data tuple on node$_1$ moves to state *Stored*. It stays in this state until the application has forwarded the data.

In the normal case, the application will forward any data tuple in the *Stored* state, and then move them to the *Forwarded* state. This instructs GeoRep to

[12] It will be back.

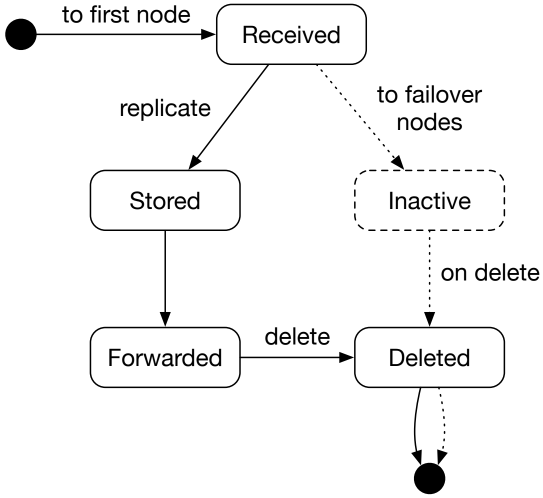


Figure 8.17: The life cycle of each data tuple on the first node.

inform the failover nodes, i.e., node_3 and node_4 in Fig. 8.15, that this data should be deleted. Finally, the data tuple is removed from the local storage in GeoRep on the first node as well.

Fig. 8.18 illustrates the cases later shown as B and C in Fig. 8.20, when a failover node discovers that all earlier nodes in the owners field no longer respond to its heartbeat requests within the stipulated timeout. It then moves the data tuple from state *Inactive* to *Stored*, and informs the application about this change. The life cycle then proceeds as above, causing the data tuple to be forwarded and then deleted on any remaining failover nodes. As described in Section 8.3.3, there is a possibility for the same data tuple to enter the *Stored* state and therefore be forwarded by multiple nodes. We do not need to create a mechanism to prevent that, as such duplication are acceptable according to our requirements.

8.2.4 Source Code

The source code, consisting of about 3500 lines of C, is publicly available[13]. This includes both the proof-of-concept implementation of the replication protocol and the test application and scripts used in the evaluations in Sections 8.4 and 8.5. ZeroMQ[14] is used for the networking layer.

[13] https://bitbucket.org/infoflexconnect/leaderlessreplication

[14] https://zeromq.org

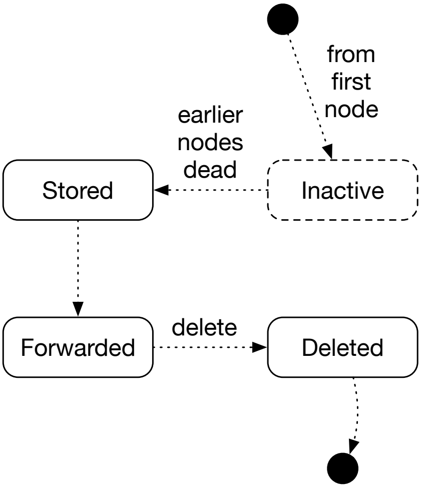


Figure 8.18: The life cycle of a data tuple in case of failover.

8.2.5 Evaluation Environment

For the evaluations later in this paper, we used a total of thirteen servers in 2021, all of them being the smallest ones offered by DigitalOcean[15] at that time: 1 GB memory, 25 GB disk, and 1 virtual x64 CPU. They all ran CentOS 7.9, with the working directory on the filesystem XFS. The code was compiled using gcc 4.8.5.

8.3 Reliability Analysis

The design of our protocol has some immediate consequences on its reliability. We will discuss these consequences next, based on the quality model ISO 25010 [38]. This model defines several characteristics for the evaluation of a software product, each one separated into several sub-characteristics. In this section we will focus on the Reliability characteristic, which contains the sub-characteristics Maturity, Availability, Fault Tolerance and Recoverability. Discussing the maturity of a new protocol does not seem meaningful, and the recoverability in terms of how GeoRep handles a lost node was already discussed in Section 8.2.1.3.

For the evaluations of the availability and fault tolerance of the proposed protocol, we will use the concepts *yield* and *harvest*, respectively, suggested

[15] `https://digitalocean.com`

by Fox and Brewer [25]. In Section 8.3.1 we discuss the availability in terms of the yield, i.e., how likely it is for a producer to be able to find a node in the GeoRep system which accepts a new data tuple. Next, in Section 8.3.2, we discuss the fault tolerance in terms of the harvest, seen as the probability that the consumer will receive at *least* one copy of each data tuple. Finally, the fault tolerance is again discussed in Section 8.3.3, now from the perspective of what happens when the communication between two or more nodes fail for some reason, and under which conditions the consumer will get at *most* one copy of a particular data tuple.

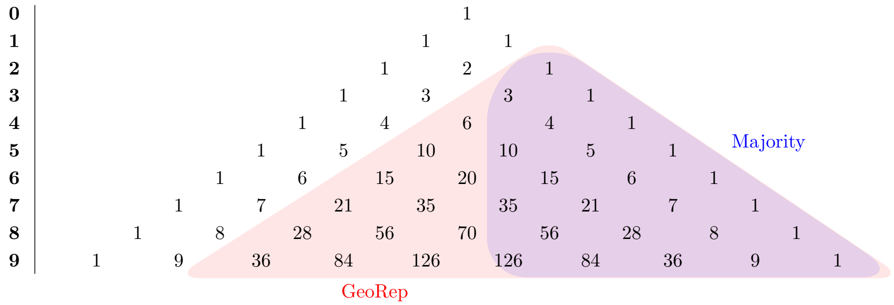
8.3.1 Availability – Yield

The *yield* [25] for GeoRep is the probability for a client to be able to find a set of at least $f+1$ (where f represents the number of nodes that are allowed to fail after data has been received and acknowledged, as discussed above) correctly functioning nodes. Here we assume that the client knows about all n nodes in the system.

To calculate this yield, we define a *node-set* as a set of nodes that can communicate with each other. Each one of n nodes is either part of, or not part of, each such set, giving a total of 2^n sets. If a node has failed, it is put in its own node-set. As we only care about sets with a size of at least 2 (i.e. $f+1$, where $f > 0$), failed nodes are automatically ignored in our calculations below. There are $\binom{n}{k}$ sets with size k. For example, consider the configuration in Fig. 8.15, where $n = 5$. The number of sets with sizes between 2 and 5 are then 10, 10, 5, and 1, respectively.

GeoRep can use all sets with a size of at least $f+1$, which for $n = 5$ and $f = 1$ there are $10+10+5+1 = 26$. In contrast, replication protocols which requires a majority of the nodes to work [61] can only use those with a size of at least $(n+1)/2$, which for $n = 5$ becomes $(5+1)/2 = 3$. There are $\binom{5}{3} + \binom{5}{4} + \binom{5}{5} = 10 + 5 + 1 = 16$ such sets. The protocols requiring fewer nodes than a majority [42, 46] for a write operation to succeed, achieve this by only allowing predefined node sets, so for n nodes there are typically only n usable node sets. For protocols replicating all data to all other nodes, only a single node set is allowed.

We illustrate the general case in Fig. 8.19, using Pascal's triangle, where the row (starting at 0, shown to the left) is the number of nodes in the system, and the values in the triangle are the number of node-sets with a particular size. The list of 1's along the left side represents the single situation where all nodes are unavailable. The next column on each row, where the value is the same as the number of nodes, represents the cases where only a single node is available. Each following column represents the cases with an increasing number of available nodes. Along the rightmost side are finally the single cases where all nodes are available.



0	1									
1	1	1								
2	1	2	1							
3	1	3	3	1						
4	1	4	6	4	1					
5	1	5	10	10	5	1				
6	1	6	15	20	15	6	1			
7	1	7	21	35	35	21	7	1		
8	1	8	28	56	70	56	28	8	1	
9	1	9	36	84	126	126	84	36	9	1

Figure 8.19: Number of node-sets usable by majority replication and GeoRep, for $f = 1$.

The node-sets usable by majority replication are the ones on the right part of Fig. 8.19. As described above, GeoRep can use not only these node-sets, but also the ones to the left except the ones in the first $f + 1$ columns.

The total number of node-sets is shown in Equation (8.8) below. The ones usable by GeoRep are then shown by Equation (8.9). The number of node-sets usable by majority replication are given by in Equations (8.10) and (8.11) for odd and even values of n, respectively. For example, going from right to left on row 3, we see that for 3 nodes we can use the single case where all nodes are available, and the 3 cases where 2 out of 3 nodes are available: $2^{(n-1)} = 2^{(3-1)} = 2^2 = 4 = 1 + 3$.

The ratio between the number of sets usable by GeoRep and the ones usable by majority replication in the best case, is then given by the expression (8.12), which simplifies to Equation (8.13). As the second term in Equation (8.14) is a polynomial, the second term in Equation (8.13) will always converge to 0, making the ratio converge to 2 for all values of f. Assuming the producer can connect to any of the system nodes, the availability is therefore up to twice as high as for other systems.

$$\begin{aligned}
\texttt{total} &= 2^n && (8.8)\\
\texttt{georep} &= 2^n - (n+1) && (8.9)\\
\texttt{majority\_odd} &= 2^{n-1} && (8.10)\\
\texttt{majority\_even} &= 2^{n-1} - \binom{n}{n/2} < \texttt{majority\_odd} && (8.11)\\
\texttt{ratio} &\geq \frac{\texttt{georep}}{\texttt{majority\_odd}} = \frac{2^n - (n+1)}{2^{n-1}} && (8.12)\\
&= 2 - \frac{n+1}{2^{n-1}} && (8.13)
\end{aligned}$$

Generally, we get:

$$\texttt{georep} = 2^n - \sum_{k=0}^{f-1} \binom{n}{k} \tag{8.14}$$

There are multiple strategies to use when selecting which node-set to use, for the situations when there are more than 1 available. The effect the selected strategy has on the system throughput is examined in Section 8.5.4.

8.3.2 Fault Tolerance – Harvest

The *harvest* [25] is the probability that each data tuple inserted into the system still exists to be output when needed. When this condition is true, the consumer will receive at least one copy of the data tuple. For GeoRep we therefore define the harvest as the probability that at least one of the nodes in the particular subset used for storing an individual data tuple is alive until the data has been forwarded to the consumer (as shown in Fig. 8.14). Again, we use concrete values, for e.g., queue and recovery times, in accordance with industry standards. According to Sahoo et al. [54], the typical lifetime of a computer system is in the order of 3–10 years. The actual mean time between failures (MTBF) for a specific system may of course be both lower and higher than this, but in the calculations below we have assumed it to be 3 years. We make no assumptions on the MTBF for other equipment in the data-center, the power grid, etc, even though those are also relevant for a full analysis.

The interval from when a data tuple is stored to when it is forwarded is typically less than one second. If a node fails exactly once every 3 years the probability that it happens in any particular second, which we denote as d_{1s}, is

$$d_{1s} = \frac{1}{3 \cdot 365 \cdot 24 \cdot 60 \cdot 60} \approx 10^{-8}$$

(assuming each second is equiprobable[16]). When the node has been repaired or replaced and then restarted, we reset the clock and assume it will run for up to 3 more years.

In our use case, an embedded system or an IoT device may send a large batch of data tuples faster than they can be fully processed. The resulting queues are typically cleared within a few hours, as the incoming traffic eventually slows down. The probability that the node that received the messages dies within this time, say 3 hours, is

$$d_{3h} = 1 - (1 - d_{1s})^{3 \cdot 60 \cdot 60} \approx 10^{-4}.$$

As the nodes are geographically distant from each other, we can further assume their failures are independent. The formula for the harvest as defined above, then simply becomes $1 - d^{f+1}$, for the relevant value of d. For the normal case when data is forwarded within a second, we get a harvest for $f = 1$ of about $1 - 10^{-8(f+1)} = 1 - 10^{-16}$, a.k.a. “16 nines”. For data that stays in the system for 3 hours, we instead get a reliability of $1 - 10^{-4(f+1)} = 1 - 10^{-8}$ for $f = 1$ and $1 - 10^{-12}$ for $f = 2$. Systems where queues are frequent might therefore want to replicate to two other nodes, but more than that is mostly just a waste of network bandwidth. Please also see Table 8.6 in Section 8.4, where only one of the nine test cases required a fourth node to be available to avoid data loss.

For replication protocols using full replication, we get a harvest of $1 - d^n$. As n grows, this of course converges more rapidly towards 1, but at the cost of significantly more data traffic and higher CPU load. We want to emphasize that as there is a possibility that all nodes fail at the same time, the harvest is never exactly 1, so data loss is always possible.

8.3.3 Fault Tolerance – Duplication Analysis

We now consider the cases that can occur in the same situation as in Section 8.2.1.2, when $n = 5$ and $f = 2$, and a message is replicated from node$_1$ to node$_3$ and node$_4$. The cases are shown in Fig. 8.20. Neither node$_2$ nor node$_5$ have seen this message, so whether they remain in contact with the other nodes has no effect here. For our SMS gateway application, the consumer here is

[16] This is of course a simplification, but we consider it to be an acceptable compromise in the interest of understandability [5].

the mobile network operator handling SMS to the recipient of each particular SMS.

A. As long as node_1 is alive, it will try to deliver the message to the consumer, and the statuses of the other nodes do not matter.

B. If node_3 concludes that node_1 is dead or for some other reason unreachable, it will adopt the message and try to deliver it. Here, the status of node_4 does not matter.

C. If node_4 loses contact with both node_1 and node_3, it will then try to deliver the message itself.

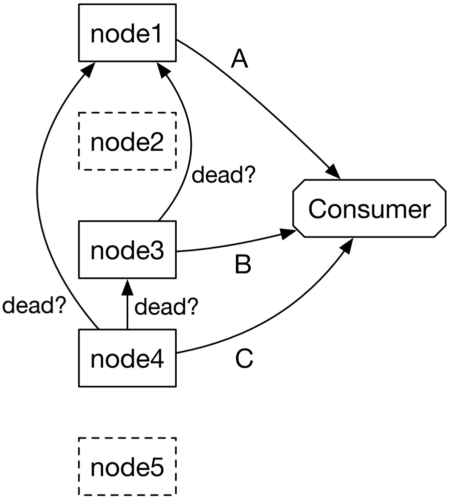


Figure 8.20: Possible duplications.

There is no way for a node to know if any of the other nodes are dead or are unreachable for another reason, e.g., being unusually slow [5, 45]. In case multiple nodes can communicate with the consumer but not with each other, messages could therefore be duplicated. We assume that the probability for this is low, and these duplications are therefore acceptable. We consider it much more likely that a lost node is dead or has lost internet connectivity entirely, and thereby also the connectivity to the consumer. In both of these two latter cases the message is delivered only once.

8.4 Failover Verification

As we see it, the most important functionality that needs verification is that data tuples inserted into the system are adopted and subsequently forwarded by another node if the original node becomes unreachable. More specifically, a data tuple should only be adopted by the first node in its *owners* list where all preceding nodes have become unreachable.

For the test case construction, we defined five different categories of nodes. At the top level we had the nodes in the *owners* list plus the *rest* of the nodes. Of the owners, we had one *originator* and a list of *failover peers*. Of those peers, we distinguished between the *first* one, the ones in the *middle*, and the *last* one. These three peer groups allowed at least one peer to have other peers before it in the *owners* list, after it, and both.

Next, we assigned a number to each category as follows, and as shown in Table 8.4: originator=1, first=2, middle=4, last=8, rest=16. Finally we created a sum of the values representing nodes that had become unavailable. As the selected values are powers of 2, this sum can be seen as a bitmask, where the bit value 0 meant the nodes in this category were still reachable, and 1 that they were not. For example, the bitmask value $00001 = 1$ meant only the originator was unreachable, and $01100 = 12$ that the originator and the first failover peers was still reachable, as well as the non-peer nodes (in the *rest* group), but not any of the other failover peers. This way we got a set of 32 unique test cases, numbered from 0 to 31, providing a reasonable coverage of possible server and network outages as each test case represented the situation where zero or more nodes in each of these categories became unavailable to all other nodes.

owners	**originator**		1
	failover peers	**first**	2
		middle	4
		last	8
rest			16

Table 8.4: The five different node categories, and their assigned bitmask values.

Of the total set of 32 possible test cases, all even numbered ones mean the originating node is still alive and reachable. Therefore no adoption should occur in any of these cases. Next, the test cases 16–31 are the same as the cases 0–15, as the reachability of nodes not in the *owners* list have no effect, regardless of how many they are. This leaves us with just 9 distinct test cases, listed in Table 8.5. We note that in cases 0 and 15, no adoption is made. In case 0, as there is no need for it, and in case 15, as there is no owner left alive to do the adoption. In case 15 there is simply an unfortunate subset of $f + 1$ nodes being unavailable, corresponding exactly to the nodes storing the tested data tuple, i.e., both the original node and all failover peers.

Number	**Unreachable**	**Adopter**	**Minimum** f
$0 = 00000$	none	none	1
1	originator	first	1
3	originator and first	middle	2
5	originator and middle	first	3
7	originator, first and middle	last	3
9	originator and last	first	3
11	originator, first, and last	middle	3
13	originator, middle and last	first	2
$15 = 01111$	all owners	none	1

Table 8.5: Relevant tests cases.

Finally, we mapped the test cases listed in Table 8.5 to concrete servers. This mapping is shown in Table 8.6, where nodes that should become unreachable are marked with *italics* and nodes that should adopt the message(s) are marked with **boldface**.

Number	**originator**	**first**	**middle**	**last**
0	node$_1$	node$_2$	node$_3$	node$_4$
1	*node*$_1$	**node**$_2$	node$_3$	node$_4$
3	*node*$_1$	*node*$_2$	**node**$_3$	node$_4$
5	*node*$_1$	**node**$_2$	*node*$_3$	node$_4$
7	*node*$_1$	*node*$_2$	*node*$_3$	**node**$_4$
9	*node*$_1$	**node**$_2$	node$_3$	*node*$_4$
11	*node*$_1$	*node*$_2$	**node**$_3$	*node*$_4$
13	*node*$_1$	**node**$_2$	*node*$_3$	*node*$_4$
15	*node*$_1$	*node*$_2$	*node*$_3$	*node*$_4$

Table 8.6: Mapping test cases to servers, marking which ones should become *unreachable* and which ones should **adopt** the replicated data tuples.

The rest of this section contains the details regarding the implementation and execution of these test cases, as well as the results.

8.4.1 Experiment Design

The critical point for a data tuple is the transfer from *Inactive* to *Stored*, shown in Fig. 8.18 in Section 8.2.3, which in turn will trigger at least one of the nodes in the *owners* list to hand the data tuple over to the application so it can ultimately be forwarded to the consumer. To simulate this sequence of events, we created a test application that performed the following steps.

1. Create a single data tuple.

2. Replicate the data tuple to all other nodes, and wait for confirmation.

3. Block all outgoing traffic from a selected subset of nodes, as specified in Table 8.6. This simulates the node having failed.

4. Wait some time to allow the blocked nodes to reach the state *Terminated* in Fig. 8.16 in Section 8.2.2, triggering the data tuple adoptions.

5. Examine the log files created on each node, to see which node or nodes adopted the data tuple.

8.4.2 Factors and Variables

For this evaluation, the only independent factor was the set of nodes which should be made unavailable, and the only dependent variable was the set of nodes adopting the data. Based on Table 8.6, all test cases in this section used $n = 4$ and $f = 3$. We also used a fixed peer order to ensure the roles of each node was predictable. Preliminary tests showed that the number of clients and messages had no effect on the behaviour, so we set both of these parameters to 1. As the adoptions were performed based entirely on local information, the concepts of recovery time, time to elect a new leader and so on, commonly evaluated for other replication protocols, were not relevant to us. The factors and variables are summarized in Table 8.7 for easy overview.

8.4.3 Execution

The tests were implemented by adding a filter between the main GeoRep logic and the ZeroMQ interface, making it possible on the application level to prevent any outgoing traffic to one or more particular other peer nodes. The shell script `run-failover.sh` was used to ensure all executions used the correct parameters, and that data was collected in the same way for all test cases.

Type	**Factor**	**Value(s)/Unit**
Independent	Disabled node(s)	None, 1, 2, 3, and/or 4
Constants	Servers, n	4
	Protection, f	3
	No of clients	1
	No of messages	1
	Separation	local
Dependent	Adopter	node number(s)
Ignored	Recovery time	seconds

Table 8.7: Experiment factors for the failover evaluation.

8.4.4 Results

Table 8.8 shows the results for each one of the test cases. For test case 0, no node was blocked, and therefore no adoptions by other nodes occurred. For the other test cases, we notice that the correct node, as specified in Table 8.6, does indeed adopt the replicated data.

No	$\mathbf{node_1}$	$\mathbf{node_2}$	$\mathbf{node_3}$	$\mathbf{node_4}$
0				
1	*blocked*	**adopts**		
3	*blocked*	*blocked* / **adopts**	**adopts**	
5	*blocked*	**adopts**	*blocked* / **adopts**	
7	*blocked*	*blocked* / **adopts**	*blocked* / **adopts**	**adopts**
9	*blocked*	**adopts**		*blocked* / **adopts**
11	*blocked*	*blocked* / **adopts**	**adopts**	*blocked* / **adopts**
13	*blocked*	**adopts**	*blocked* / **adopts**	*blocked* / **adopts**
15	*blocked*	*blocked* / **adopts**	*blocked* / **adopts**	*blocked* / **adopts**

Table 8.8: Failover results, showing *blocked* nodes and the ones **adopting** any data tuples.

Except for node$_1$, all blocked nodes also adopt the replicated data tuples. The reason for this is that as they are blocked, they never get any life signs from the other nodes and therefore must consider these too to be unreachable. As discussed in Section 8.3.3, this would however rarely lead to any data duplications.

8.5 Throughput Evaluation

For an evaluation of the proposed protocol primarily focused on quality attributes, we designed a controlled experiment [51]. The overall goal was to evaluate the throughput in a few different configurations.

8.5.1 Experiment Design

We used a sequence of tasks corresponding with the queue related operations performed by the type of systems described as our system model in Section 8.1.1, resulting in realistic experiments. We created a test application which itself created the messages, and discarded them when all tasks described below were completed.

1. A new message was stored locally and replicated according to the selected configuration. The application waited for acknowledgements from the others servers before returning control to the application.

2. A message was extracted from the queue.

3. The extracted message was deleted from all servers where it was stored.

A benchmark suite commonly used for evaluating replication systems is the Yahoo! Cloud Serving Benchmark (YCSB) [17]. Using the same suite makes it easy to compare different solutions, but as it is designed for web server type systems and not store-and-forward systems, YCSB was not meaningful for us.

8.5.2 Factors and Variables

In addition to the usual Independent and Dependent factors, we found it relevant to describe the independent factors that we set to constant values, and the dependent factors which we chose to ignore. These are all described in more detail below, and summarized in Table 8.9.

8.5.2.1 Independent Factors

The primary factors in these experiments were selected to give a deeper understanding of the behaviour under different circumstances.

The number of servers was varied from 2 to 7. The number of client connections was varied between 1 and 1000. For clarity, only subsets of these intervals are shown in the diagrams below.

Type	Factor	Value(s)/Unit
Independent	Servers, n	$2 \ldots 7$
	Clients	$1, 3, 10, \ldots, 1000$
	Separation	Local, Geographical
Constant	Protection, f	1
	Transient	5 s
	Steady-state	30 s
Dependent	Throughput	MPS
	Min RTT	Microseconds, μs
Ignored	Recovering	MPS
	Duplications	Ratio

Table 8.9: Experiment factors.

We used servers both within the same data center and in multiple time zones. This way we could examine the effect the physical distances between the servers, and thereby the different round-trip times, had on the system throughput. The data centers used for the different numbers of servers, are shown in Table 8.10. The idea was to keep the sites as geographically separated as possible. Only when using 6 or 7 servers did we use data centers relatively close to each other.

Data center	Number of servers					
	2	3	4	5	6	7
Amsterdam	✓	✓	✓	✓	✓	✓
New York	✓	✓	✓	✓	✓	✓
San Francisco		✓	✓	✓	✓	✓
Bangalore			✓	✓	✓	✓
Singapore				✓	✓	✓
London					✓	✓
Toronto						✓

Table 8.10: Data centers used for the Geographical cases.

The reliability of the power and internet infrastructure is also relevant, but these factors mainly affect the availability of the system, not its fault tolerance. We get high availability by having a large number of possible node sets, and as we saw in Fig. 8.19 in Section 8.3.1, the most effective way to increase the number of such sets is to increase the number of nodes, n. This value is already selected as one of the independent factors.

8.5.2.2 Constants

We motivate setting the protection f to 1 by recalling the discussion about reliability in Section 8.3.2. For normal operations, where messages are forwarded within the same second as they were received, even setting f to such a low value as 1 gives a reliability of about $1 - 10^{-16}$.

All configurations were tested for 35 seconds. First, there was a transient phase of 5 seconds, allowing the CPU caches and TCP parameters to stabilize. Next, the application continued to run in the steady-state phase for another 30 seconds.

8.5.2.3 Dependent/Response Variables

For all configurations, i.e. the combinations of one particular value for each of the independent variables, the response variable of most interest to us in this experiment was the total system throughput. This throughput was defined as the number of messages processed per second (MPS), according to the sequence of tasks described in Section 8.5.1.

We also measured the minimum RTT between each pair of nodes. The median round-trip time would be more relevant for answering the question of what a typical response time would be. However, as discussed in Section 8.1, we are more interested in the system resilience, achieved by replicating the data tuples to nodes at some minimum physical distance from each other. A large RTT clearly is no guarantee that the nodes are far apart, but due to the finite speed of light, a small RTT requires the nodes to be near each other.

8.5.2.4 Ignored Response Variables

Other response variables that might be of interest mainly concern the behaviour when a failed server is detected, and the time-span afterwards during which the system is reassigning messages to new servers.

8.5.3 Execution

Before each test, all servers were reset to a known empty starting state. The files for local storage were removed, so they could be recreated as needed. The application was then started on all servers, with the selected values for the independent variables provided as command line parameters.

The test application counted the number of messages processed each second by each server, values that were then summarized into a result for the full system. Finally, the median of the values for each of the 30 seconds in the steady-state phase was calculated.

8.5.4 Results

Here we present a summary of the results from our throughput evaluations, made to establish an initial intuition of how this protocol behaves. As mentioned, we varied the number of servers up to 7, and the number of clients up to 1000, even though the diagrams just show the results for representative subsets.

In a local network, the total system throughput increased with the number of nodes up to 40 437 MPS on 7 nodes with 300 clients, shown in Fig. 8.21. The minimum RTT varied between 143 μs and 420 μs.

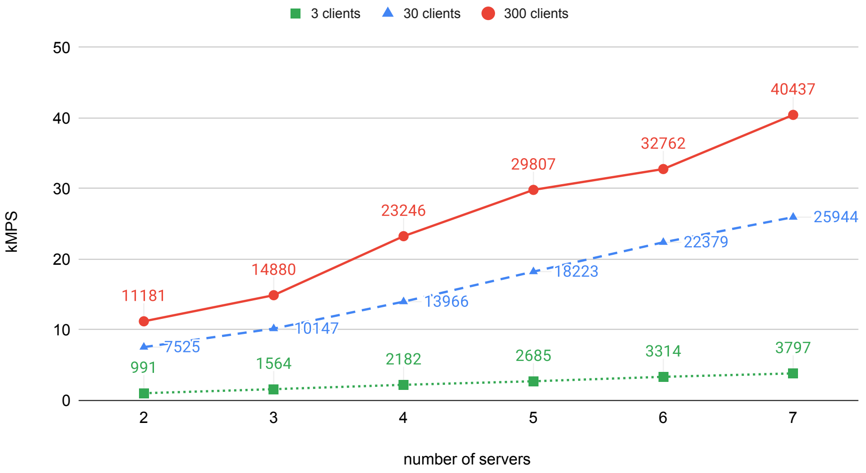


Figure 8.21: System throughput as a function of the number of servers, all running in the same data center.

When GeoRep was deployed in a cluster of geo-separated servers, throughput again increased with the number of nodes. The peak throughput levels were much lower than in the local case, due to the longer round-trip times. For the same reason, the system spent more time waiting for responses, lowering the CPU load. This allowed us to increase the number of clients to 1000. Fig. 8.22 shows how the throughput reached 9048 MPS for 2 nodes and 24 085 MPS for 7 nodes.

In Fig. 8.23 we see the performance hit resulting from the replication logic. The entries for $f = 0$ show the case when not using any replication at all. We also ran a few tests using $f = 2$. Other than occasional heartbeat traffic, the executed program code in GeoRep is just a very thin layer on top of LevelDB. As expected, the throughput scales almost linearly by the number of nodes, around 35–40 kMPS per node.

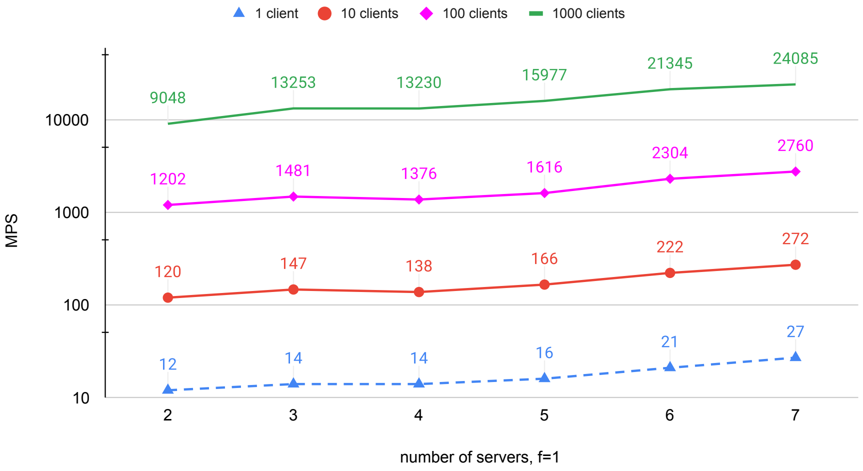


Figure 8.22: System throughput as a function of the number of servers, running in different data centers on multiple continents. Please note that the Y axis is logarithmic, to match the logarithmic increase in the number of clients.

For 3 geo-separated nodes, the minimum RTT averaged 105 ms. For 7 nodes, the relatively distant nodes in Bangalore and Singapore resulted in an increase to 138 ms. Fig. 8.24 shows the RTT between a few selected pairs of nodes. For example, the RTT from Toronto (in column 3) is quite low to New York, almost the same to San Francisco and Amsterdam, and quite high to Bangalore. The profiles for nodes geographically close to each other, e.g., New York and Toronto, are notably similar.

Based on Fig. 8.24, we saw that instead of replicating messages to a random selection of nodes, we could select the f ones with the smallest RTT from where the message was received, ignoring nodes with an RTT lower than some predefined limit, say 10 ms. This minimum value ensures messages are always replicated outside of the critical region mentioned in Section 8.1.

We set the number of servers to 7, and varied the number of clients between 100 and 1000. We varied the minimum RTT limit between 1, 20, and 100 ms, based on the following reasoning. A minimum of 1 ms prevents a node from replicating to another node within the same data center. This level protects from local internet and power outages. The RTT between New York

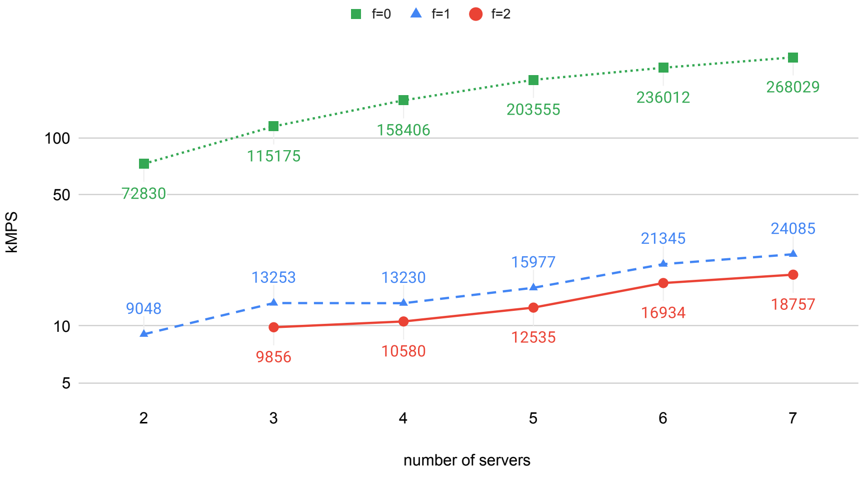


Figure 8.23: System throughput as a function of the number of servers, running in different data centers on multiple continents, when varying f between 0, 1, and 2. The number of clients is 1000.

and Toronto, and between the nodes in Europe, is around 10 ms. By setting a minimum of 20 ms, these nodes must find peers further away, such as the one in California or one across the Atlantic. This level protects from larger outages covering bigger areas. When increasing the limit to 100 ms, we also prevent replication within the American continent and between the American east coast and Europe. The data tuples are then always replicated at least about one third of the total circumference of the earth. Increasing the limit further would not have any practical application. With a larger number of nodes in more parts of the world, other RTT limits would be meaningful, offering a larger number of tradeoff points between throughput and reliability. The achieved throughput for the three tested cases are shown in Fig. 8.25.

8.5.5 Comparative Evaluation

To get a performance comparison between GeoRep and Paxos, we used the C implementation LibPaxos3[17]. Based on the requirements described in Section 8.1.3, we assumed that a full implementation based on Paxos would need to do at least two operations per message. First, the data would be added to the replicated event list, including the *owners* field described in Section 8.2.1. As only the node first in the *owners* field would be allowed to forward the

[17] `https://bitbucket.org/sciascid/libpaxos`

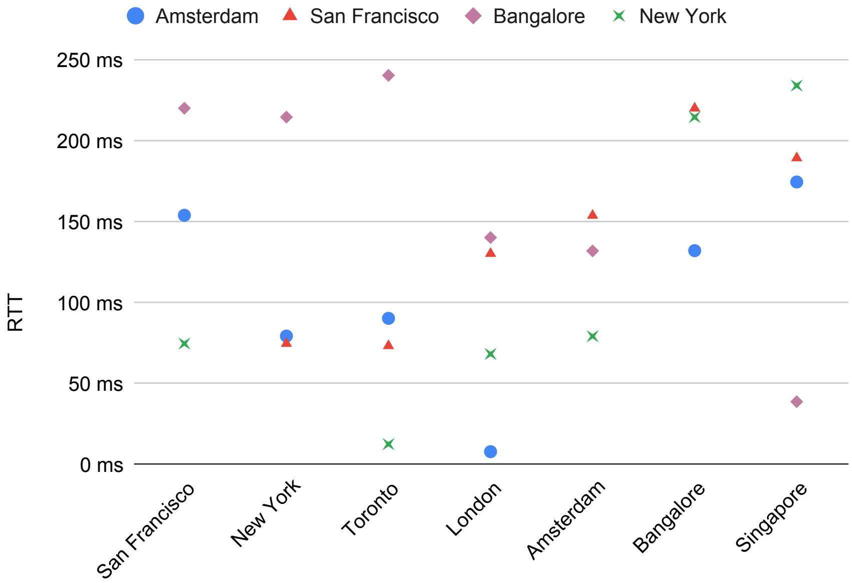


Figure 8.24: Round-trip time (RTT) for various pairs of servers.

message, we avoid duplications. When the correct node has forwarded the message, the message id would be replicated again, with a flag marking it as being delivered. We can therefore get the number of messages that could be processed by a Paxos based solution per second, by simply counting the number of events we can submit and divide by 2.

The set of reachable nodes would be stored within the event log as well, providing a consensus on when the failover logic should be activated. There still exists at least one sequence of events where a message may be duplicated, described below. To the best of our knowledge, this situation can not be completely avoided, as any process may crash between promising to do something and then doing it, or between doing something and then informing that it has been done. However, we already stated in Section 8.1.2 that a limited number of message duplications, caused by situations like these, are acceptable.

1. A node N finds itself being the owner of a particular message m.

2. Node N sends m.

3. Node N replicates the event that m has been forwarded. Before this event has been sent, N crashes.

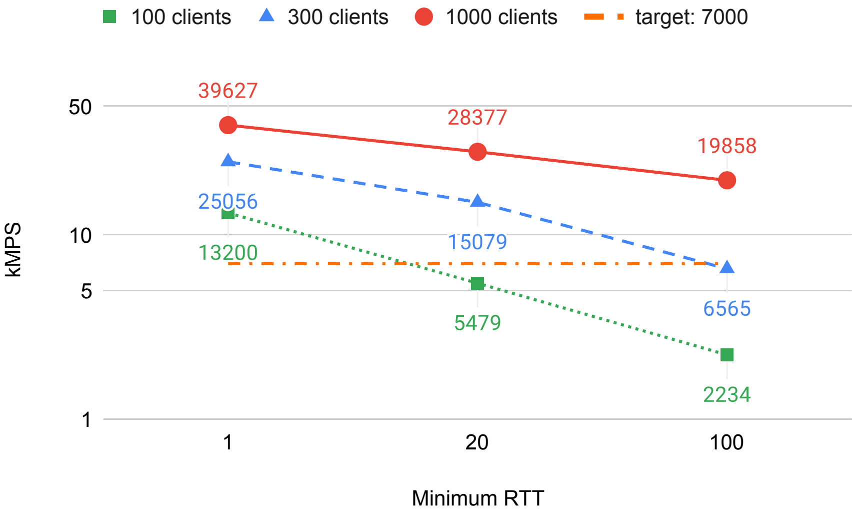


Figure 8.25: System throughput for various minimum RTT limits. In this experiment we use 7 nodes, giving a target throughput of $7 * 1000 = 7000$ MPS.

4. The remaining nodes discover that N no longer responds, and after a consensus round m is adopted by the next node in its *owners* list.

We tested the Paxos implementation in the same environments as GeoRep, first with up to 7 servers in the same data-center, and then on up to 7 geo-separated servers. The numbers when all nodes are within the same data-center, in Table 8.11 on the line marked *Local*, should be compared to the ones for GeoRep in Fig. 8.21. We see that for 3 nodes Paxos is faster than GeoRep, even when GeoRep has 300 parallel client threads. However, while the system throughput increases when nodes are added in GeoRep, the throughput instead decreases in Paxos. We then compare the numbers for the geo-separated configurations to the ones for GeoRep in Fig. 8.22. Paxos is now more on par with GeoRep for 10 parallel clients. Just as in the previous configuration, the clear performance increase seen for GeoRep is not present with Paxos. The number of clients had no measurable effect in this experiment.

The main advantage with a Paxos based solution is that the risk for duplicated messages would be 0, due to the stricter reliance on consensus in Paxos. With up to at least 7 nodes running within the same data-center, we also get at least 1000 MPS per node, our target as specified in Section 8.1.3. Paxos is not as suitable in geo-separated configurations, nor provides the clear scale-up for more servers as seen with GeoRep.

	Number of servers				
	3	**4**	**5**	**6**	**7**
Local/Paxos	22827	13366	16021	13798	9343
Local/GeoRep	14880	23246	29807	32762	40437
Separated/Paxos	756	356	217	211	243
Separated/GeoRep	13253	13230	15977	21345	24085

Table 8.11: LibPaxos3 system throughput, in messages per second (MPS).

8.6 Discussion

In our experiments, the proposed protocol was shown to be able to leverage the ordering independence of the data tuples and thereby perform better as the number of clients, and thereby also the number of parallel requests, increased. As shown in Fig. 8.25 in Section 8.5, the highest recorded throughput for the geo-distributed case was 28 377 MPS when using 7 servers with a minimum RTT of at least 20 ms between each other, or sufficiently far apart to avoid having more than 1 server fail due to a single power or network outage. The independence between the data tuples enables us to reach much more than our target 1000 MPS per node, as long as there are sufficiently many clients.

8.6.1 Threats to Validity

The identified validity threats are grouped [16, 39] for better overview.

8.6.1.1 Construct

The validity threat “construct” concerns whether the experiment measures the right thing. Differences in hardware, programming language, the number of clients, servers, and replication groups, as well as selected test scenarios make it difficult to compare absolute numbers to previous work. The failover mechanism uses only local operations, and the rate of this was not measured.

8.6.1.2 Internal

Internal validity threats concern the causal relationship between two variables. Even though an existing system was the driving force for the requirements addressed by GeoRep, a new and minimal application was written for these experiments. This avoided the threat of any confounding variables introduced by the existing implementation and simplified the reproducibility.

In a production environment, the client applications will of course not run on the same machine as GeoRep. Separating them will result in more time passing for the client, between submitting a data tuple for replication, and getting the confirmation back. On the other hand, it will leave more CPU to GeoRep, possibly increasing its performance for the CPU bound parts.

To address the threat of additional confounding factors, all cases were run for a relatively long time. As we focused on the median, any temporary variances in the environment were effectively filtered out.

8.6.1.3 External

External validity threats concern whether the results are still valid in a more general context. Due to not having a coordinating server, our proposal is only usable for situations where the stored elements have no relative order. Applications where this is true, other than in our embedded systems use case, are email gateways. These gateways also route messages from companies to their customers, but instead of delivering messages to network operators, they are delivered to email servers and ultimately to the customers' mailboxes. Here too, the relative order between messages does not matter, there are no reliable end-to-end acknowledgements,[18] and each message is important to its recipient. Here, the quality requirements for these systems also mean the system must provide high availability to the senders, and as messages must not get lost despite temporary failures of both system nodes and recipient systems.

8.7 Related Work

8.7.1 Replication in Practice

Among others, Helland and Campbell in 2009 [32] and Hellerstein and Alvaro in 2019 [33], argued that shifting the focus from the storage layer up to application semantics may lead to better solutions. In our case, this shift enabled us to not only take advantage of the lower network requirements by partial replication, but also to lower the network usage even further by avoiding the cost of maintaining a total order of the messages. It also made it possible, in case of a network partition, to let other subsets than the one containing the majority of the original nodes continue working, thereby making the system available to the senders in the minority group(s).

[18] A common workaround for emails are tracking pixels, but these are usually possible to disable on the client side. Some email services, e.g., `hey.com`, see them as a threat to privacy and explicitly blocks them.

8.7.2 Replication Protocols

Other store-and-forward systems are application-to-application message queues, e.g. Apache Kafka [41]. In Apache Kafka the data in the system can be spread over multiple subsets of the nodes, with each such subset being called a partition. A partition has an elected leader, which handles all reads and writes, and zero or more replicas which are kept in sync using a very efficient mechanism. Should the leader become unavailable, one of the replicas takes its place. This gives an automatic ordering of the events, but at the cost of being sensitive to the network latency between the client and the replica leader. GeoRep avoids this cost, as it has no leader. Instead, clients are free to connect to any node of their choice, thereby minimizing the latency time and as a result maybe also maximizing the throughput. It is quite likely that a Kafka-based solution would perform well in the same environment as used in our tests. It would however not satisfy our "minimal changes to an existing application" requirement from Section 8.1.3.

For systems where a global ordering must be maintained, e.g., fast atomic multicast [15] and white-box atomic multicast [27], the replication protocols are often based on a variant of Paxos [43] or Raft [50]. The Paxos variant Mencius [47] was designed to perform well even in wide-area networks with high inter-node latency. One of the ways they achieve this is by using a multi-master setup, where the leadership is divided among all nodes similarly to GeoRep. However, as all data is sent to all other nodes, the throughput does not increase when nodes are added to the system. These systems would also require a consensus round among all nodes when each message has been processed and can be deleted, while GeoRep only needs to send this information to the f nodes involved in the replication for that particular message. As is shown in the evaluations of both white-box atomic multicast [27] and Mencius [47], reducing the number of communication steps has a clear and positive effect on the system performance. We do not need the higher consistency these protocols provide, so we can reduce the number of communication steps even further. The experiment in Section 8.5.5 showed some of these differences in practice.

Another solution would be to store the data tuples in an SQL database, where there are plenty of replication methods. However, as SQL databases must maintain the ACID (Atomicity, Consistency, Isolation, and Durability) [30] properties of the data, those methods work best within a local server cluster. With geo-separated servers, the higher round-trip times cause a significant performance degradation in our case, as the "find and remove the next data tuple" operation would require a global, synchronous lock. Preliminary tests with such a configuration resulted in a throughput in the order of 1 message per second. Comparing GeoRep with an SQL database in this paper would therefore not be meaningful.

8.8 Conclusions and Future Work

With the purpose of increasing the resilience of a store-and-forward system, we designed a solution based on application semantics instead of lower level storage operations. Several approaches to data replication exist, but we could not find any existing solutions with sufficiently high throughput for geo-separated configurations. Our main contribution in this work is the description and implementation of a new protocol, based on partial replication. When deployed on 7 nodes running on different continents, it provided a total throughput of 24 085 messages per second, almost 100 times higher than a comparable implementation based on Paxos. The primary trade-off is that during a network outage, there is a small risk for message duplication.

Naturally, we welcome replication studies of our protocol. The experiments can be varied along several different dimensions, e.g., a) using other programming languages than C, b) using other frameworks than ZeroMQ, c) using a larger number of nodes, d) separating the client applications into separate nodes, and e) considering other use cases and application areas. The source code used in the experiment is open sourced to facilitate such studies.

There is no consensus among the nodes regarding the reachability of the other nodes, so the number of use cases for the failover verification in Section 8.4 is actually higher than 9, and increases with higher values of f. A deeper analysis to find the exact formula for which of these test cases involving the reachabilities from multiple nodes can actually occur, their expected outcome, and comparing this with the actual behaviour, would be interesting, but is left as future work.

For predictable disasters [49], e.g., hurricanes, floods and tsunamis, it should be possible to temporarily disable some servers beforehand as replication targets, to minimize data loss. The same strategy could even be used for more unpredictable disasters causing power failures, in those cases triggered by the affected nodes switching to battery power.

Bibliography

[1] G. Aceto, A. Botta, P. Marchetta, V. Persico, and A. Pescapé. A comprehensive survey on internet outages. *Journal of Network and Computer Applications*, 113(2018):36–63, jul 2018.

[2] M. K. Aguilera and R. E. Strom. Efficient Atomic Broadcast Using Deterministic Merge. In *Proceedings of the Symposium on Principles of Distributed Computing (PODC)*. ACM, 2000.

[3] M. Ahamad and M. Ammar. Performance Characterization of Quorum-Consensus Algorithms for Replicated Data. *IEEE Transactions on Software Engineering*, 15(4):492–496, apr 1989.

[4] S. Almeida, J. Leitão, and L. Rodrigues. ChainReaction: a Causal+ Consistent Datastore based on Chain Replication. In *Proceedings of The European Professional Society on Computer Systems (EuroSys)*. ACM, 2013.

[5] P. A. Alsberg, G. G. Belford, S. R. Bunch, J. D. Day, E. Grapa, D. C. Healy, E. J. McCauley, and D. A. Willcox. Research in Network Data Management and Resource Sharing, Synchronization and Deadlock. Technical Report CCTC-WAD Document Number 6508, Center for Advanced Computation, University of Illinois, 1977.

[6] P. A. Alsberg and J. D. Day. A Principle for Resilient Sharing of Distributed Resources. In *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE Computer Society Press, 1976.

[7] H. Attiya, A. Bar-Noy, and D. Dolev. Sharing memory robustly in message-passing systems. *Journal of the ACM*, 42(1):124–142, 1995.

[8] P. Bailis and K. Kingsbury. The Network is Reliable. *Communications of the ACM*, 57(9):48–55, sep 2014.

[9] N. Belaramani, M. Dahlin, L. Gao, A. Nayate, A. Venkataramani, P. Yalagandula, and J. Zheng. PRACTI replication. In *Proceedings of the Conference on Networked Systems Design & Implementation (NSDI)*. USENIX, 2006.

[10] D. Brahneborg, W. Afzal, A. Cauševic, and M. Björkman. Superlinear and Bandwidth Friendly Geo-replication for Store-and-forward Systems. In *Proceedings of the International Conference on Software Technologies (ICSOFT)*. INSTICC, 2020.

[11] S. Braun and S. Desloch. A Classification of Replicated Data for the Design of Eventually Consistent Domain Models. In *International Conference on Software Architecture Companion*, ICSA-C. IEEE, 2020.

[12] M. Bravo, L. Rodrigues, and P. Van Roy. Saturn: A Distributed Metadata Service for Causal Consistency. In *Proceedings of the European Conference on Computer Systems (EuroSys)*, 2017.

[13] Y. Cheng, M. T. Gardner, J. Li, R. May, D. Medhi, and J. P. Sterbenz. Analysing GeoPath diversity and improving routing performance in optical networks. *Computer Networks*, 82:50–67, 2015.

[14] P. Coelho and F. Pedone. Geographic State Machine Replication. Technical Report USI-INF-TR-2017-3, Faculty of Informatics Università della Svizzera italiana Lugano, Switzerland, 2017.

[15] P. R. Coelho, N. Schiper, and F. Pedone. Fast Atomic Multicast. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2017.

[16] T. D. Cook and D. T. Campbell. *Quasi-experimentation: Design and Analysis for Field Settings*. Rand McNally, Chicago, 1979.

[17] C. Cooper, T. Radzik, N. Rivera, and T. Shiraga. Benchmarking cloud serving systems with YCSB. In *Proceedings of the Symposium on Cloud Computing (SoCC)*, SoCC. ACM, 2010.

[18] M. Dahlin, B. B. V. Chandra, L. Gao, and A. Nayate. End-to-end WAN Service Availability. *IEEE/ACM Transactions on Networking*, 11(2):300–313, 2003.

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

[20] J. Du, C. Iorgulescu, A. Roy, and W. Zwaenepoel. GentleRain: Cheap and Scalable Causal Consistency with Physical Clocks. *Proceedings of the Symposium on Cloud Computing (SOCC)*, 2014.

[21] Edsger Wybe Dijkstra. Co-operating Sequential Processes. In *Programming languages*. Academic Press Inc, 1968.

[22] P. T. Eugster, P. A. Felber, R. Guerraoui, and A.-M. Kermarrec. The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, 35(2):114–131, 2003.

[23] M. J. Fischer and A. Michael. Sacrificing Serializability to Attain High Availability of Data in an Unreliable Network. In *Proceedings of the SIGACT-SIGMOD Symposium on Principles of Database Systems (PODS)*. ACM, 1982.

[24] P. Fouto, J. Leitão, and N. Preguiça. Practical and Fast Causal Consistent Partial Geo-replication. In *Proceedings of the International Symposium on Network Computing and Applications (NCA)*. IEEE, 2018.

[25] A. Fox and E. A. Brewer. Harvest, Yield, and Scalable Tolerant Systems. In *Proceedings of the Workshop on Hot Topics in Operating Systems (HOTOS)*. IEEE, 1999.

[26] P. Gill, N. Jain, and N. Nagappan. Understanding Network Failures in Data Centers: Measurement, Analysis, and Implications. In *Proceedings of the SIGCOMM Conference*, New York, NY, USA, 2011. ACM.

[27] A. Gotsman, A. Lefort, and G. Chockler. White-box Atomic Multicast. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2019.

[28] R. Guerraoui and A. Schiper. Genuine Atomic Multicast in Asynchronous Distributed Systems. *Theoretical Computer Science*, 254(1-2):297–316, 2001.

[29] N. Gunther, P. Puglia, and K. Tomasette. Hadoop superlinear scalability. *Queue*, 13:20–42, 5 2015.

[30] T. Haerder and A. Reuter. Principles of transaction-oriented database recovery. *ACM Computing Surveys*, 15(4):287–317, 1983.

[31] C. Hale. You can't sacrifice partition tolerance. `https://codahale.com/you-cant-sacrifice-partition-tolerance`, 2010 (Retrieved May 2020).

[32] P. Helland and D. Campbell. Building on Quicksand. In *Proceedings of the Conference on Innovative Data Systems Research (CIDR)*. ACM, 2009.

[33] J. M. Hellerstein and P. Alvaro. Keeping calm: when distributed consistency is easy. *arXiv preprint arXiv:1901.01930*, 2019.

[34] J. M. Hellerstein and P. Alvaro. Keeping calm. *Communications of the ACM*, 63, 8 2020.

[35] H. Howard and R. Mortier. Paxos vs Raft: Have We Reached Consensus on Distributed Consensus? In *Proceedings of the Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC)*, 2020.

[36] D. Hutchison and J. P. Sterbenz. Architecture and design for resilient networked systems. *Computer Communications*, 131:13–21, 10 2018.

[37] F. Iqbal and F. A. Kuipers. Disjoint paths in networks. *Wiley Encyclopedia of Electrical and Electronics Engineering*, pages 1–11, 1999.

[38] ISO/IEC. ISO/IEC 25010. `https://iso25000.com/index.php/en/iso-25000-standards/iso-25010`, 2020. Accessed 2020-06-07.

[39] A. Jedlitschka, M. Ciolkowski, and D. Pfahl. Reporting experiments in software engineering. In *Guide to advanced empirical software engineering*. Springer, London, 2008.

[40] P. R. Johnson and R. H. Thomas. RFC 677: The Maintenance of Duplicate Databases, 1975.

[41] J. Kreps, N. Narkhede, and J. Rao. Kafka: a Distributed Messaging System for Log Processing. In *Proceedings of the SIGMOD Workshop on Networking Meets Databases (NetDB)*, Athens, Greece, 2011.

[42] A. Kumar. Hierarchical Quorum Consensus: A New Algorithm for Managing Replicated Data. *IEEE Transactions on Computers*, 40(9):996–1004, 1991.

[43] L. Lamport. The Part-time Parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.

[44] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3), 1982.

[45] B. G. Lindsay, P. G. Selinger, C. Galtieri, J. N. Gray, R. A. Lorie, T. G. Price, F. Putzolu, I. L. Traiger, and B. W. Wade. Notes on distributed databases. Technical Report Report RJ2571 (33471), 1979.

[46] M. Maekawa. A $\sqrt{N}$ Algorithm for Mutual Exclusion in Decentralized Systems. *ACM Transactions on Computer Systems (TOCS)*, 3(2):145–159, 1985.

[47] Y. Mao, F. P. Junqueira, and K. Marzullo. Mencius: Building Efficient Replicated State Machines for WANs. In *Proceedings of the USENIX Conference Operating System Design and Implementation (OSDI)*, 2008.

[48] A. Mauthe, D. Hutchison, E. K. Cetinkaya, I. Ganchev, J. Rak, J. P. Sterbenz, M. Gunkelk, P. Smith, and T. Gomes. Disaster-resilient communication networks: Principles and best practices. In *International Workshop on Resilient Networks Design and Modeling*, RNDM. IEEE, 2016.

[49] B. Mukherjee, M. F. Habib, and F. Dikbiyik. Network adaptability from disaster disruptions and cascading failures. *IEEE Communications Magazine*, 52(5):230–238, 2014.

[50] D. Ongaro and J. Ousterhout. In Search of an Understandable Consensus Algorithm. Technical report, Stanford University, 2014.

[51] C. Robson and K. McCartan. *Real world research.* John Wiley & Sons, 2016.

[52] J. P. Rohrer, A. Jabbar, and J. P. Sterbenz. Path diversification for future internet end-to-end resilience and survivability. *Telecommunication Systems*, 56(1):49–67, 2014.

[53] J. B. Rothnie and N. Goodman. A Survey of Research and Development in Distributed Database Management. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 1977.

[54] S. S. Sahoo, B. Ranjbar, and A. Kumar. Reliability-aware resource management in multi-/many-core systems: A perspective paper. *Journal of Low Power Electronics and Applications*, 11(1):7, 2021.

[55] N. Schiper and F. Pedone. On the Inherent Cost of Atomic Broadcast and Multicast in Wide Area Networks. In *Proceedings of the International Conference on Distributed Computing and Networking (ICDCN).* Springer Berlin, Heidelberg, 2008.

[56] N. Schiper, P. Sutra, and F. Pedone. P-store: Genuine Partial Replication in Wide Area Networks. In *Proceedings of the Symposium on Reliable Distributed Systems (SRDS).* IEEE, 2010.

[57] M. Shapiro, N. Pregui, C. Baquero, and M. Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types. Technical Report RR-7506, Inria – Centre Paris-Rocquencourt, 2011.

[58] J. P. Sterbenz and D. HJutchison. ResiliNets Wiki. `https://resilinets.org`, 2016 (accessed July 26, 2021).

[59] M. Stonebraker and E. Neuhold. A Distributed Data Base Version of Ingres. Technical Report ERL-M612, California University, Berkeley., 1976.

[60] D. B. Terry, M. M. Theimer, K. Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. Managing update conflicts in bayou, a weakly connected replicated storage system. *SIGOPS Oper. Syst. Rev.*, 29(5):172–182, Dec. 1995.

[61] R. H. Thomas. A majority consensus approach to concurrency control for multiple copy databases. *ACM Transactions on Database Systems (TODS)*, 4(2):180–209, 1979.

[62] B. Vass, J. Tapolcai, D. Hay, J. Oostenbrink, and F. Kuipers. How to model and enumerate geographically correlated failure events in communication networks. In *Guide to Disaster-Resilient Communication Networks*, pages 87–115. Springer, Cham, 2020.

[63] M. Yousif. Cloud Computing Reliability—Failure is an Option. *IEEE Cloud Computing*, 5(3):4–5, may 2018.

Paper E.
Resilient Conflict-free Replicated Data Types without Atomic Broadcast

Resilient Conflict-free Replicated Data Types without Atomic Broadcast. Daniel Brahneborg, Wasif Afzal, Saad Mubeen.

In *Proceedings of the International Conference on Software Technologies (ICSOFT)*. INSTICC, 2022.

Abstract

In a distributed system, applications can perform both reads and updates without costly synchronous network round-trips by using Conflict-free Replicated Data Types (CRDTs). Most CRDTs are based on some variant of atomic broadcast, as that enables them to support causal dependencies between updates of multiple objects. However, the overhead of this atomic broadcast is unnecessary in systems handling only independent CRDT objects. We identified a set of use cases for tracking resource usage where there is a need for a replication mechanism with less complexity and network usage as compared to using atomic broadcast. In this paper, we present the design of such a replication protocol that efficiently leverages the commutativity of CRDTs. The proposed protocol CReDiT (CRDT enhanced with intelligence) uses up to four communication steps per update, but these steps can be batched as needed. It uses network resources only when updates need to be communicated. Furthermore, it is less sensitive to server failures than current state-of-the-art solutions as other nodes can use new values already after the first communication step, instead of after two or more.

9.1 Introduction

Many distributed systems need to efficiently manage external resources. These resources could be, e.g., network traffic, the number of times to show a specific web advertisement, and more. In this work, we will consider an application with one or more users, each one paying for the resources they use. Each user has a credit balance, representing payments made and resources used. This balance is then used as basis for their next invoice. The system clearly must take great care in maintaining these credit balances.

Regardless of how reliable modern computer components have become, occasional server outages are unavoidable [1, 5]. In order to make the *service* available despite these *server* outages, we need multiple servers [12, 24, 25, 31], preferably independent and geographically separated [13]. The challenge is to maintain accurate records of the resource consumption across all these servers.

Unfortunately, maintaining consistency in a distributed system can easily lead to decreased performance [14], and in the presence of network partitions, fully distributed consistency and high availability simply cannot coexist [10, 16]. An interesting exception was identified by Alsberg and Day [4], suggesting what is basically a precursor to Conflict-free Replicated Data Types (CRDTs) [27]:

> *"An example [of a specific exception] is an inventory system where only increments and decrements to data fields are permitted and where instantaneous consistency of the data base is not a requirement."*

CRDTs have become popular for distributed systems over the past few years, partly because of their convenient property of having the same value regardless of the order of the operations performed on them. When instantaneous consistency is not required, local operations can be performed on them immediately, without the need for time-consuming network round-trips. The new state is instead regularly broadcast to the other nodes. All nodes therefore eventually get the same value for the object. There are two main groups of CRDTs:

State-based CRDTs send their full state [11] between the nodes. This makes them immune to both packet loss and packet duplications, but it can quickly lead to excessive network usage for data types with a large state, and to massive storage requirements when there is a high number of clients [2]. A special case of these are **delta-based CRDTs**, which only transmit the part of the state changed by local updates [3, 15].

Operation-based CRDTs send only the individual operations [6]. These typically use less network resources, but require reliable delivery where all operations are successfully received by all nodes exactly once [32].

Even if the operation order on a single CRDT object does not matter, many applications update an object based on the values of another. In order to enforce such causal dependencies, most CRDT implementations use reliable causal broadcast (RCB) where all nodes get the same set of packets in more or less the same order [8, 26]. RCB is typically based on atomic broadcast, which can ensure not only that all packets are delivered in the same order, but also that this happens only if all nodes are still reachable. A simple way to implement atomic broadcast is Skeen's algorithm, which requires a set of network packets sent back to the sender, and then a third set of "commit" [17] packets to all destinations from which the sender got the reply. When the causality check is based only on Lamport clocks [20] this can give false positives, in turn leading to unnecessary network traffic and delays [7].

The purpose of this work is to find a replication protocol for state-based CRDTs not having any causal dependencies, where the replication uses less network resources than previously proposed solutions. We use user credits as the motivating example, typically implemented as CRDT PN-counters [27]. In short, a PN-counter is a set of pairs of integers $\mathbb{Z}$, merged by the operation `max()`, where the integers are used for positive and negative changes, respectively. Its value is the difference between these two integers. We refer to the paper by Shapiro et al. for a more detailed description.

Our proposed protocol *CReDiT* (CRDT enhanced with intelligence) extends state-based CRDTs by augmenting the local state with additional information in order to avoid unnecessary network traffic. All data is periodically resent until it has been acknowledged by each other node, making the protocol immune to occasional packet loss.

We will describe this work using Shaw's framework [28], which categorizes research in three different ways. First is the research setting, which is what kind of research question or hypothesis is being addressed. Our setting is *Methods/Means*, described in Section 9.2. Next is the research approach. Here the desired result is a new *Technique*, described in detail in Section 9.3. The third way is the result validation, which is done in Section 9.4. We discuss the results in Section 9.5, present related work in Section 9.6, and end the paper with our conclusions in Section 9.7.

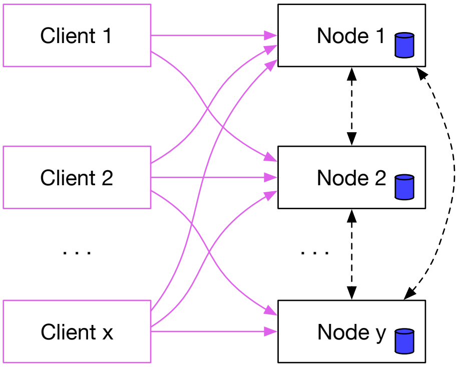


Figure 9.26: We have x clients connected to y nodes, which in turn are all connected to each other. Each node maintains its own database of the credit balance for each client.

9.2 Method

In Shaw's framework [28], the purpose of a "Methods/Means" setting is to find an answer to a research question such as "what is a better way to accomplish X". After defining our system model in Section 9.2.1, we will therefore define our "X" in Section 9.2.2, and what we mean by "better" in Section 9.2.3.

9.2.1 System Model

We assume that we have a distributed system of independently running nodes, communicating over an asynchronous network. The physical network can use any topology, as long as there is at least one logical path between each pair of nodes. We further assume fair-lossy links, i.e., packets may be dropped, but if a packet is sent infinitely often it will eventually be received. Packets may also be duplicated and received out of order. The nodes have local memory and stable storage, and can recover after crashing. We also assume there are some number of clients, each one connecting to any node or nodes. As the clients send requests to a node, their resource counter in that node is updated. Figure 9.26 shows the situation with x clients and y nodes. This work addresses the communication between the nodes, shown with dashed lines.

9.2.2 Functional Requirements

The functionality we need, i.e. our "X", matches almost exactly what Almeida and Baquero [2] call *eventually consistent distributed counters* (ECDCs). These use the *increment* operation for updating the counter, and *fetch* for reading its current value. *Fetch* returns the sum of updates made. A second call to *fetch* returns the previous value plus any locally made updates since the previous call. Eventually, *fetch* will return the same value on all nodes, i.e., the above named sum of updates. In addition to an ECDC, we also allow negative updates, which means we can count both the resources used and the payments made.

9.2.3 Quality Requirements

We also need to specify our quality requirements, i.e. what we mean by "better". We base these on ISO 25010 [19], a taxonomy which puts quality attributes into eight different groups of characteristics, each one divided into a handful of sub-characteristics. The latter are written below in the form *Main characteristic / Sub-characteristic*.

The CAP theorem [10, 16] tells us that given a network partition, we cannot have both data consistency and availability. With the ISO 25010 nomenclature, this means we need to choose between *Functional Suitability / Functional Correctness* (the needed degree of precision) and *Reliability / Availability*. We strongly prioritize the latter, as it is usually a good business decision to let customers keep using your service, even when facing the risk of occasional overdrafts.

For the *Performance Efficiency / Capacity*, we assume the system has up to about 10 nodes, and that there are up to 1000 clients using its resources. For now, we do not address the remaining quality characteristics in ISO 25010 [19].

Assuming that all clients are independent, we can model the time between each update for each client using the exponential distribution. This distribution has the probability density function $f(x) = \lambda e^{-\lambda x}$, and the cumulative distribution function (CDF) $P(X < x) = 1 - e^{-\lambda x}$. In both functions λ is the inverse of the client specific mean interval μ, and x is the length of the interval.

This CDF has an interesting property, as it is always less than 1. In other words, there will always exist a time interval of length x without any updates. If x is measured in seconds, we will have an alternating sequence of some number of seconds with updates, and some other number of seconds without.

9.3 Proposed Technique

Our proposed protocol is based on PN-counters [27], augmented with data to keep track of the values on the other nodes. This data, and how it is used, is described in Section 9.3.1. Section 9.3.2 describes the protocol as a state

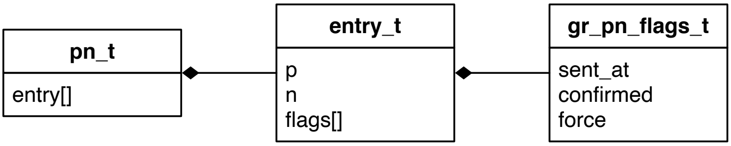


Figure 9.27: The three data types used by CReDiT.

machine, and Section 9.3.3 shows a sample scenario in a system with two nodes. The protocol is named *CReDiT*, inspired by its basis in CRDT.

9.3.1 Prototype Implementation

We assume the application has some sort of collection of resource counters. For the network communication, CReDiT uses a separate transport layer. Each counter, named `pn_t` in our implementation, contains a map from node identifiers to instances of the structure `entry_t`. An `entry_t` contains the two fields `p` and `n`, just as the original PN-counters.

We extend the `entry_t` structure with a map from node identifiers to `gr_pn_flags_t` structures, containing the fields `sent_at`, `confirmed`, and `force`, described below. These fields are therefore specific for each pair of nodes. The three types are shown in Figure 9.27.

`sent_at`: timestamp
This is the most recent time the current value was sent to the other node. In our implementation, for simplicity but without any loss of generality, we use a resolution of one second for this field.

`confirmed`: boolean
This is set when the incoming values from another node are identical to what is stored locally, so we know that we do not need to send the same data to that node again.

`force`: boolean
This is set when a value must be sent on the next flush, overriding the `confirmed` flag.

In the function descriptions below, we use `A` for the local node where the code is executed, `B` for one of the remote nodes, `entry` for the instance of the `entry_t` structure on `A`, `x` for a random node, and `*` to designate all nodes. The protocol uses the functions listed below, of which only `flush()` and `receive()` perform any network operations. We have marked the original PN-counter functionality with “**PN**” and our additions with “**New**”.

`init(x, p, n)`

This is used when loading values from external storage on startup.

PN: It sets `entry[x].p` and `entry[x].n` to the supplied values.

New: It clears `entry[x].flags[*]`.

`update(delta)`

This is called to update the resource counter.

PN: If the delta is positive, `entry[A].p` is increased with `delta`, and if it is negative, `entry[A].n` is increased with `-delta`.

New: As we know that node `A` is the only one updating the `entry[A]` fields, no other node has these exact values now, and we can therefore clear the `entry[A].flags[*].confirmed` flags.

`fetch()`

This function is called by the application to get the current value of the counter.

PN: It returns the sum of all `entry[*].p` fields minus the sum of all `entry[*].n` fields.

`flush()`

This should be called regularly by the application, in order to initiate the replication.

New: Entries are sent if the `force` flag is set, or if the `confirmed` flag is not set and `sent_at` is different than the current time. Afterwards, `sent_at` is set to the current time, and `force` is set to false. The use of `sent_at` here allows the application layer to call this function as often as it wants. In contrast to solutions based on atomic broadcast, CReDiT does not wait for any replies from the other nodes.

`receive(B, x, p, n)`

This function is called by the transport layer, when new data has been received from node `B` concerning values on node `x`.

PN: The field `entry[x].p` is updated to the maximum of its current value and the incoming `p`, and similarly for `entry[x].n` and `n`.

New: If `entry[x].flags[B].confirmed` is set, `entry[x].flags[B].force` is set. If `entry[x].p` or `entry[x].n` was changed, `entry[x].flags[B].force` is also set and `entry[x].flags[*].confirmed` are cleared. The `entry[x].flags[B].confirmed` flag is always set though, as we know that node B has these particular values. Finally a callback is made to the application, which can now persist the new data. This persisted data is what the application should provide to the init() function after being restarted.

Our implementation was based on GeoRep [9], which supplied networking code and configuration management for keeping track of the nodes to which data should be replicated.

9.3.2 State Machine

Figure 9.28 shows a compact summary of the algorithms and the effects of the flags. There is a separate state machine for each individual counter, and for all pairs of nodes. The state of each machine is an effect of the node specific flags in entry_t: If the force flag is set or if the current time differs from the value of the sent_at attribute, the machine is in state 1 or 3. If the confirmed flag is set, it is in state 2 or 3. Each counter starts at the filled black circle, and immediately goes to state 0. This represents the case when both force and confirmed are false. In all states, update() and receive() update the corresponding (p, n) pair(s). All functions described in Section 9.3.1 can be called in any of these four states, but functions not affecting a particular state are omitted for clarity.

9.3.3 Data Flow

In a system with the two nodes A and B, these are the steps taken when A updates a shared counter.

1. A updates the value of a new counter with +2. This creates the counter, and A sets p=2 and n=0 in entry[A].

2. After at most one second, A moves B to state 1. On the next call to flush() from the application layer, the values for A are sent to B, after which A sets entry[B].sent_at to the current time.

3. When B receives this data, it stores A's values p=2 and n=0, and sets the flags confirmed and force in entry[A].

4. As A has sent_at=now for B and force is not true, any additional calls to flush() will not cause more data to be sent to B.

5. B has force set to true for A, so the next time flush() is called on B, the (p=2, n=0) pair for A is sent back to A.

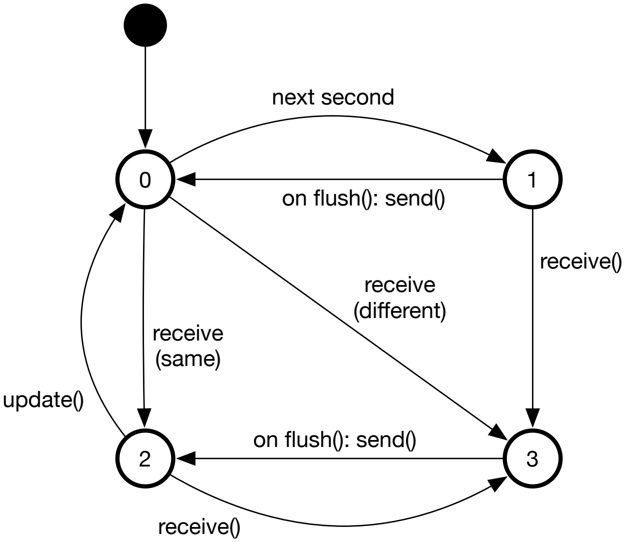


Figure 9.28: States on a node. Each pair of nodes has its own state. The black circle is the starting point. For the states 1 and 3, the `force` flag is set. For the states 2 and 3, the `confirmed` flag is set.

6. Next, A gets the (p=2, n=0) pair for A from B. As these are the same values it already has, it sets `confirmed` to `true` for B. It does not set `force`. After this, both A and B has `confirmed` set to `true` for each other, and agree on the (p=2, n=0) pair. No more data is sent by either side.

If the data sent in step 2 is lost, A will obviously not get this data back from B. When `flush()` is called during the next second or later, A will see the missing `confirmed` flag and send the data again. This way, the `confirmed` flag on node A prevents repeated transmissions of the same data from A to B. As we assume fair-lossy links as mentioned in Section 9.2.1, B will eventually receive this data.

If the data from B to A in step 5 is lost, B will still have the `confirmed` flag set, so it will not send the data again. However, A will not have this flag set, so it will send the data to B again. From B's perspective, as the `confirmed` flag for this entry is set, A and B should already have the same data. Hence, as B sees the same data again from A, it can deduce that A's `confirmed` flag is not set. B can fix that by setting its `force` flag for A, causing the data to be sent back to A on the next call to `flush()`. This way, the `force` flag on node B prevents repeated transmissions of the same data from node A to B.

The combined effect of the `confirmed` and `force` flags is that any data packet can be lost, and the protocol will still recover. Once all nodes have the same set of confirmed values, no more data will be sent until after the next call to `update()`.

9.4 Evaluation

Here we discuss the validation of the proposed protocol regarding its functionality, correctness, and scalability. As both `update()` and `fetch()` only work on local data, the availability is 100% by construction.

9.4.1 Functional Validation

We already know that CRDTs in general, and PN-counters in particular, converge on the same value on all nodes, thanks to the broadcast and merge mechanisms [27] also used by CReDiT. We therefore only need to show that the new fields do not invalidate this. For `sent_at` this is obvious, as this field only limits how often data is sent.

The `confirmed` flag prevents data being sent from node A to node B, when B has shown that it already has the exact same values as A. As long as this is true, sending this data again is of no use to anyone. When the values on A change, its `confirmed` flags are cleared, giving the original CRDT behaviour. If the values on B change, this field is cleared on B, causing the data to be broadcast to all nodes, including A, which in turn clears the field for the other nodes, also getting us back to the original CRDT behaviour.

As described in Section 9.3.3, the `force` flag handles the case when the sent values returned to the sender (A) are lost. As long as the values on A are unchanged, A would otherwise keep sending them to B because no confirmation is received. For new values sent by A, B would notice the update and send it back to A, just as for any other CRDT.

9.4.2 Correctness Conditions

A state-based CRDT ensures that all updates originating on a particular node can never be done in a different order on another node, as its current state always includes its previous state. Its commutativity further ensures that even if the relative order of updates made on different nodes may differ between the nodes, the value of a CRDT object will eventually be the same on all nodes. As this order may differ between nodes, we do not get *serializability* [23].

Whether we get linearizability [18] is not entirely clear. Herlihy and Wing states that the “real-time precedence ordering of operations” should be respected. This is indeed the case on each particular node. However, in a distributed system with nodes A and B we can have a sequence such as the following.

1. A stores the value 1 in variable x.
2. A stores the value 2 in variable x.
3. B reads the value of variable x.
4. B reads the value of variable x.

The data replication from node A to node B may be initiated both after step 1 and 2. Furthermore, the new data may arrive to node B both before and after step 3, as well as after step 4. Node B can therefore see both the values 1, 2, or something else. Still, if node B would read the value 2 in step 3, we can guarantee that step 4 will not read the value 1 (a.k.a. *monotonic reads*). Also, if node A would read the value of variable x, after step 1 it would get 1, and after step 2 it would get 2 (a.k.a. *read your writes*).

9.4.3 Scalability

The memory usage for each counter is $\mathcal{O}(n)$ for the values, and $\mathcal{O}(n^2)$ for the flags. There is no transaction log as for operation-based CRDTs, so for a given n the memory requirement is constant.

For an update on node A, up to 4 sets of network packets are triggered. After these steps, all n nodes will have the same values, as well as knowing that the other $n-1$ nodes have them too. Because of this knowledge, no more data is sent until the next update is made.

1. Node A sends the new (p, n) pair to the other $n-1$ nodes.
2. After receiving the new pair, these $n-1$ nodes send back their updated values.
3. For a system with 3 or more nodes, the $n-1$ nodes has at least one set of `flags` where `confirmed` is not set. So, `flush()` on these nodes will broadcast the updated values to the remaining $n-2$ nodes.
4. If a packet in the previous set is received from a node y on a node x before x has broadcast the update itself, the `force` flag will be set on node x, causing the value to again be sent from node x to node y.

An update will therefore cause a total of up to $(n-1)+(n-1)+(n-1)(n-2)+(n-1)(n-2)=2(n-1)^2$ network packets to be sent in the system. This quadratic scale-up makes this protocol unsuitable for systems with a large number of nodes, even though the decision for when this is true must be done on a case by case basis.

The packet size will be proportional to the number of updated counters since the last confirmation, but it is not affected by the number of updates to a particular counter. The number of updates also has no effect on the number of required network packets, making the quadratic scale-up less of a problem than it may seem.

Additionally, counters with no updates on a particular node, after its `confirmed` flag is set, stay in state 2 in the state machine shown in Figure 9.28. In this state `flush()` generates no network traffic.

9.4.4 Real-world Evaluation

There are a couple of seemingly obvious measurements that could be used in order to evaluate protocol behaviour in real-world situations. First, we could measure the number of function calls per second. However, as all functions either just modify local data structures or are asynchronous, this would effectively only measure the CPU speed of our test machines. Second, we could measure the time from when `flush()` is called until the data has reached all other nodes. Unfortunately, this would just measure the round-trip time between the nodes. Third, we could compare some performance aspect of the application that originally triggered this work. Currently, the best solution for that application appears to be using a replicated MySQL server. However, we have not found any way to do the required multi-master replication with geo-separated nodes, while still getting acceptable performance (of at least 1000 updates per second).

Instead, we will compare our protocol with PN-counters based on atomic broadcast. In particular, we have observed that for counters with updated data, most algorithms for atomic broadcast use fewer communication steps than CReDiT does. For counters without updates, CReDiT uses fewer. So, we want to measure the relative frequency between these two cases. From two production systems running the motivating application mentioned above, we retrieved sample log files containing the time stamps of events that would trigger an update of one of our counters.

The first file covers an interval of 91 hours in the middle of September 2021, with a total of 78 987 events. Within this interval we observed the occurrence of events during each hour, but only during 3358 out of a total of 5460 minutes, and during 35 166 out of the total of 327 600 seconds. Despite an average of 0.241 events per second, there is an event only during 10.7% of the seconds in this interval. The second file covers 6 hours in August 2021, during which there were 328 948 events, an average of 11.4 events per second. Still, there was at least one event during only 28357 of the included 28800 seconds (98.5%).

We do not have enough data points to find the most fitting statistical distribution for the events handled by the application, but it seems to be one of the uneven ones, e.g. the exponential distribution discussed in Section 9.2.3. The periods without any updates, where CReDiT is maximally efficient, are therefore more frequent than one perhaps would expect.

9.5 Discussion

According to Urbán et al. [30], having a designated sequencer serializing all operations in the system, uses the fewest number of communication steps per message, namely 2. The trade-off cost to achieve this is that the sequencer node needs much outgoing network bandwidth as it does a broadcast of all messages to all other nodes. Most other atomic broadcast protocols need more communication steps, but let each node broadcast its own messages.

As we saw in Section 9.4.3, our proposed protocol performs worse than this in both aspects, as it requires up to 4 communication steps and that all nodes broadcast all updated values. When there are no updates, our protocol instead does not communicate at all.

The round-trip times between each pair of nodes has little or no effect on this protocol, for several reasons. First, the updated data can be flushed at any interval, which just has to be longer than the maximum round-trip time. By default, this interval is therefore 1 second. Second, as the data sent is the full new state of each counter, the number of updates between each flush does not affect the amount of data sent. Third, as new data is immediately available to each node after being received, a temporary delay on one link between two nodes only affects those two specific nodes. This improves the reliability of the system, as updated values sent just before a crash can be used by the other nodes immediately after being received.

The increased storage requirements caused by our adding new data fields is well compensated for by the elimination of chatter on the network during quiet periods.

9.6 Related Work

Almeida et al. [3] address a problem similar to ours. Their δ-CRDTs support both duplicated network packets, just as state-based CRDTs do, and achieving the lower bandwidth requirements of operation-based CRDTs. Their anti-entropy algorithm (corresponding to our `flush()`) sends only the part of the state affected by local operations performed on the current node. For a CRDT with a large total state this δ-state is typically smaller than the full state replicated by state-based CRDTs.

One way to ensure that all servers have the same data is to use a replication protocol which can "guarantee that service requests are executed in the same order at all resource sites" [4]. The most common solution to this problem is to model the system as a replicated state machine, using a variant of Paxos [21] or Raft [22]. For the counters we need, the request order does not matter. The implementation complexity and network bandwidth required by these protocols are therefore unnecessary.

Almeida and Baquero [2] defined Eventually Consistent Distributed Counters (ECDC), which is the same partition tolerant abstraction addressed in our work. Their solution, called *Handoff Counters*, also works well over unreliable networks. Their counters aggregate the values in a few central nodes, making them scale better according to the number of servers than our solution does. By creating a map of these counters, they would provide a reasonable solution for our resource counting. However, the aggregation is rather complex, consisting of a 4-way handshake and 9 data fields.

Skrzypczak et al. [29] also addressed the synchronization overhead of state machine replication by using a single network round-trip for updates and not having a leader. To get linearizability, they coordinate using the query operations, with repeated round-trips until the returned values stabilize. In contrast, our protocol can accept both updates and queries during all types of network partitions, and can respond to queries without doing any additional round-trips.

9.7 Conclusions

Generally, layered architectures are good, reducing the complexity of each individual layer. In the case of building state-based CRDTs on top of atomic broadcast, we saw that the resulting system may use unnecessary network resources. By instead taking advantage of the lack of causality between the operations of our CRDT counters, we were able to design a new protocol with lower network requirements during periods without updates. The described approach can be used with any state-based CRDT, as long as it is possible to determine if the incoming values differ from the local values.

Acknowledgments

This work was sponsored by The Knowledge Foundation industrial PhD school ITS ESS-H and Braxo AB.

Bibliography

[1] G. Aceto, A. Botta, P. Marchetta, V. Persico, and A. Pescapé. A comprehensive survey on internet outages. *Journal of Network and Computer Applications*, 113(2018):36–63, jul 2018.

[2] P. S. Almeida and C. Baquero. Scalable Eventually Consistent Counters over Unreliable Networks. *Distributed Computing*, 32:69–89, 2019.

[3] P. S. Almeida, A. Shoker, and C. Baquero. Delta state replicated data types. *Journal of Parallel and Distributed Computing*, 111:162–173, 2018.

[4] P. A. Alsberg and J. D. Day. A Principle for Resilient Sharing of Distributed Resources. In *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE Computer Society Press, 1976.

[5] P. Bailis and K. Kingsbury. The Network is Reliable. *Communications of the ACM*, 57(9):48–55, sep 2014.

[6] C. Baquero, P. S. Almeida, and A. Shoker. Pure operation-based replicated data types. *arXiv preprint arXiv:1710.04469*, 2017.

[7] J. Bauwens and E. G. Boix. Improving the Reactivity of Pure Operation-Based CRDTs. In *Proceedings of the Workshop on Principles and Practice of Consistency for Distributed Data (PaPoC)*, 2021.

[8] K. P. Birman and T. A. Joseph. Reliable Communication in the Presence of Failures. *ACM Transactions on Computer Systems*, 5(1):47–76, Jan. 1987.

[9] D. Brahneborg, W. Afzal, A. Cauševic, and M. Björkman. Superlinear and Bandwidth Friendly Geo-replication for Store-and-forward Systems. In *Proceedings of the International Conference on Software Technologies (ICSOFT)*. INSTICC, 2020.

[10] E. A. Brewer. Towards Robust Distributed Systems. In *Proceedings of the Symposium on the Principles Of Distributed Computing (PODC)*. ACM, 2000.

[11] C. F. Carlos Baquero, Paulo Sérgio Almeida, Alcino Cunha. Composition in State-based Replicated Data Types. *Bulletin of EATCS*, 3(123), 2017.

[12] Y. Cheng, M. T. Gardner, J. Li, R. May, D. Medhi, and J. P. Sterbenz. Analysing GeoPath diversity and improving routing performance in optical networks. *Computer Networks*, 82:50–67, 2015.

[13] M. Dahlin, B. B. V. Chandra, L. Gao, and A. Nayate. End-to-end WAN Service Availability. *IEEE/ACM Transactions on Networking*, 11(2):300–313, 2003.

[14] D. Didona, P. Fatourou, R. Guerraoui, J. Wang, and W. Zwaenepoel. Distributed Transactional Systems Cannot Be Fast. In *Proceedings of the Symposium on Parallelism in Algorithms and Architectures (SPAA)*, New York, NY, USA, 2019. ACM Press.

[15] V. Enes, P. S. Almeida, C. Baquero, and J. Leitao. Efficient Synchronization of State-based CRDTs. In *Proceedings of the International Conference on Data Engineering (ICDE)*. IEEE Computer Society, 2019.

[16] S. Gilbert and N. A. Lynch. Brewer's Conjecture and the Feasibility of Consistent, Available, Partition-Tolerant Web Services. In *Proceedings of the Symposium on the Principles Of Distributed Computing (PODC)*, 2004.

[17] A. Gotsman, A. Lefort, and G. Chockler. White-box Atomic Multicast. In *Proceedings of the International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2019.

[18] M. P. Herlihy and J. M. Wing. Linearizability: a Correctness Condition for Concurrent Objects. *ACM Transactions on Programming Languages and Systems*, 12(3):463–492, jul 1990.

[19] ISO/IEC. ISO/IEC 25010. `https://iso25000.com/index.php/en/iso-25000-standards/iso-25010`, 2020. Accessed 2020-06-07.

[20] L. Lamport. Time, Clocks, and the Ordering of Events in a Distributed System. *Communications of the ACM*, 21(7):558–565, 1978.

[21] L. Lamport. The Part-time Parliament. *ACM Transactions on Computer Systems*, 16(2):133–169, 1998.

[22] D. Ongaro and J. Ousterhout. In Search of an Understandable Consensus Algorithm. Technical report, Stanford University, 2014.

[23] C. H. Papadimitriou. The serializability of concurrent database updates. *Journal of the ACM (JACM)*, 26(4):631–653, 1979.

[24] J. P. Rohrer, A. Jabbar, and J. P. Sterbenz. Path diversification for future internet end-to-end resilience and survivability. *Telecommunication Systems*, 56(1):49–67, 2014.

[25] J. B. Rothnie and N. Goodman. A Survey of Research and Development in Distributed Database Management. In *Proceedings of the International Conference on Very Large Data Bases (VLDB)*, 1977.

[26] F. B. Schneider, D. Gries, and R. D. Schlichting. Fault-tolerant broadcasts. *Science of Computer Programming*, 4(1):1–15, 1984.

[27] M. Shapiro, N. Pregui, C. Baquero, and M. Zawirski. A comprehensive study of Convergent and Commutative Replicated Data Types. Technical Report RR-7506, Inria – Centre Paris-Rocquencourt, 2011.

[28] M. Shaw. The Coming-of-Age of Software Architecture Research. In *Proceedings of the International Conference on Software Engineering (ICSE)*. IEEE Computing Society, 2001.

[29] J. Skrzypczak, F. Schintke, and T. Schütt. Linearizable State Machine Replication of State-based CRDTs without Logs. In *Proceedings of the Symposium on Principles of Distributed Computing (PODC)*. ACM, 2019.

[30] P. Urbán, X. Défago, and A. Schiper. Contention-Aware Metrics for Distributed Algorithms: Comparison of Atomic Broadcast Algorithms. In *Proceedings of the International Conference on Computer Communications and Networks (IC3N)*. IEEE, 2000.

[31] B. Vass, J. Tapolcai, D. Hay, J. Oostenbrink, and F. Kuipers. How to model and enumerate geographically correlated failure events in communication networks. In *Guide to Disaster-Resilient Communication Networks*, pages 87–115. Springer, Cham, 2020.

[32] G. Younes, A. Shoker, P. S. Almeida, and C. Baquero. Integration Challenges of Pure Operation-Based CRDTs in Redis. In *Proceedings of the Workshop on Programming Models and Languages for Distributed Computing (PMLDC)*, New York, NY, USA, 2016. ACM.

Index